Les exceptions en JAVA

Exercice 1 — *Prise en compte des exceptions dans l'agence bancaire*

1. Faire le bilan des cas d'erreurs possibles (cas de la méthode débiter)
2. Proposer pour chaque cas une classe d'exception spécifique.
3. Isoler dans le code les conditions d'erreur et proposer un traitement pour chacune d'elles.
4. Tester cette nouvelle version de l'application.

Exercice 2 — *Dates et exceptions*

Soit la classe `Date` suivante :

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

  public Date(int j, int m, int a) {
    jour = j;
    mois = m;
    annee = a;
  }
}
```

(1) Modifier le constructeur de cette classe de manière à ce qu'il renvoie une exception de type `DateException` si le jour ou le mois ne correspond pas à une date valide (on supposera que les mois ont tous 30 jours). Pour cela, vous définirez une classe `DateException` qui contient deux constructeurs, un sans argument et un avec un argument de type `String`.

(2) Modifier le code suivant, de manière à ce que cette exception soit traitée :

```
public static void main(String[] args) {
  Date d = new Date(32, 14, 2001);
}
```

(3) 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` respectivement, et on adaptera le code pour les prendre correctement en compte. En particulier, il sera nécessaire de procéder à une capture multiple dans le `main`. (On utilisera les méthodes `getMessage` et `printStackTrace` de la classe `Exception`.)

(4) Enfin, on souhaite prendre en compte le nombre exact de jours dans chaque mois. Pour cela, on déclarera une sous-classe de la classe `JourException`, qu'on appellera `JourDansMoisException`. On adaptera ensuite le code, de façon à ce qu'une exception de type `DateException` soit lancée et traitée si le jour est 32 ou plus et le mois 13 ou plus, à ce qu'une exception de type `JourException` soit lancée et traitée si le jour est 32 ou plus, à ce qu'une exception de type `MoisException` soit lancée et traitée si le mois est 13 ou plus, et à ce qu'une exception de type `JourDansMoisException` soit lancée et traitée si le jour n'existe pas dans le mois considéré (par exemple, le 31 avril, ou le 30 février). Pour simplifier, on supposera que le mois de février a toujours 28 jours.

Exercice 3 — *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. Pour tester le code qui sera écrit, on vous demande dans un premier temps de créer un fichier texte contenant les dix premiers entiers (un par ligne), écrits en toutes lettres ("un", "deux", "trois", etc.), puis un second similaire contenant les 10 entiers suivants ("onze", "douze", etc.).

On veut écrire un programme dont le fonctionnement est le suivant : il doit demander à l'utilisateur de saisir un entier i, et il lira ensuite la $i$ème ligne du fichier texte dont le nom est passé en paramètre au moment du lancement du programme. Dans le cas où tout s'est bien passé, il affichera le contenu de la ligne lue ; dans le cas contraire, il informera l'utilisateur de l'exception qui a été déclenchée. Néanmoins, on supposera qu'une des lignes (dont le numéro sera tiré au hasard) est inaccessible à l'utilisateur, et que vouloir afficher son contenu génèrera donc une exception, de type `LigneException`.

On commencera donc par définir une classe `LigneException`, qui sera écrite sur le même modèle que `DateException`, ainsi qu'une classe `Fichier` qui permettra de lire les informations du fichier dont le nom apparaîtra en variable d'instance (de type `String`).

Cette classe contiendra une méthode `lire`, qui prendra en paramètre le numéro d'une ligne, et renverra le contenu de cette ligne (sous la forme d'une variable de type `String`). Elle lèvera également une exception de type `LigneException` si le numéro de la ligne passé en paramètre correspond à un entier interdit, tiré au hasard entre 1 et 10, ou bien une exception de type `IOException` si le fichier contient moins de lignes que le numéro de ligne passé en paramètre.

Voici les différentes étapes à réaliser :

1. Définir la classe `LigneException`, sur le même modèle que la classe `DateException`.
2. Définir la classe `Fichier`, dont le constructeur prend en paramètre le nom du fichier (variable de type `String`), pour initialiser la valeur de la variable d'instance correspondante.
3. Définir la méthode d'instance `lire` de cette classe, décrite ci-dessus. On peut lire dans un fichier (ici, de nom "toto.txt"), par l'intermédiaire d'une instance de la classe `BufferedReader` :

   ```
   BufferedReader input = new BufferedReader(new FileReader("toto.txt"));
   ```

   Pour lire chacune des lignes du fichier (la lecture se fait ainsi de façon séquentielle), on peut alors utiliser la méthode `readLine()`, qui renvoie une variable de type `String` :

   ```
   String ligne = input.readLine();
   ```

   Cette méthode retourne `null` si la fin de fichier est atteinte (plus aucune ligne à lire). Enfin, il faut penser à fermer le fichier, une fois la lecture effectuée :

   ```
   input.close();
   ```

   Pour générer un nombre aléatoire entre 1 et 10, on pourra utiliser une instance de la classe `Random` :

   ```
   int nombreAleatoire = (new Random()).nextInt(10)+1;
   ```

   Attention : la classe `Random` se trouve dans le package `java.util`, et les classes `BufferedReader`, `FileReader` et `IOException` dans le package `java.io`.
4. Enfin, écrire le `main`. Il faut créer une instance de fichier (dont le nom est celui passé en paramètre au moment de l'appel au programme), puis utiliser la classe `Scanner` pour faire saisir à l'utilisateur un numéro de ligne (entier) entre 1 et 10. On affichera alors la valeur de la ligne associée dans le fichier considéré. Évidemment, il faudra gérer à l'aide d'un bloc `try`–`catch` (plusieurs `catch`) les exceptions qui peuvent ainsi être générées : si la ligne saisie est inaccessible à l'utilisateur, si le fichier comporte moins de lignes que le numéro de ligne saisi ou n'existe pas (`FileNotFoundException`), ou bien toute autre exception (dont il faudra au préalable déterminer la ou les source(s) potentielle(s)).

   Attention : la classe `FileNotFoundException` se trouve dans le package `java.io`, et la classe `Scanner` dans le package `java.util`.
5. (Bonus.) Modifier le code de façon à gérer la fermeture du fichier (à l'aide de `input.close()`) par l'intermédiaire d'un `finally`, sans modifier la gestion des exceptions dans le `main`.