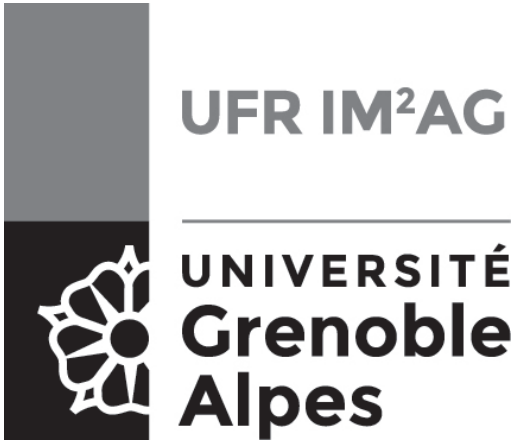

DU-ISN
Utilisation de fichiers en java
gestion d'erreurs via les exceptions

Accès aux fichiers

Découpé en trois phases

- ouverture du fichier
  - demande au système de retrouver un emplacement sur le disque à partir du nom d'un fichier
  - il vérifie au passage que l'accès est légitime
  - renvoie un descripteur de fichier qui stocke la position courante
- accès au fichier via le descripteur qui peut être :
  - `InputStream` $\Rightarrow$ `Scanner` ou `DataInputStream`
  - `OutputStream` $\Rightarrow$ `PrintStream` ou `DataOutputStream`
- fermeture du fichier via `close` ou à la destruction du descripteur

Descripteurs déjà ouverts par le système

- `System.in` : `InputStream` permettant de lire au clavier
- `System.out` : `PrintStream` permettant d'écrire à l'écran
- `System.err` : `PrintStream` permettant d'écrire à l'écran

Lire et écrire

Lecture avec un `Scanner`

- construit à partir d'un `InputStream` (flux brut d'octets)
- lit plusieurs caractères en avance pour reconstituer des
  - mot (`next`)
  - lignes (`nextLine`)
  - entiers (`nextInt`)
  - ...
- ne pas construire plusieurs `Scanners` à partir d'un seul `InputStream`

Écriture avec un `PrintStream`

- spécialisation d'un `OutputStream`
- contient des méthodes comme `print`/`println`
- répercute les écritures ligne par ligne

En cas d'erreur...

Une exception est levée

```
InputStream in =
    new FileInputStream("Toto.txt");
```

Provoque l'arrêt du programme et un affichage de l'erreur

```
Exception :
java.io.FileNotFoundException: Toto.txt (No such file or directory)
Trace de la pile
java.io.FileNotFoundException: Toto.txt (No such file or directory)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at ExceptionLectureSimple.main(ExceptionLectureSimple.java:9)
```

Pour gérer l'erreur

La gestion d'une erreur se fait en attrapant l'exception correspondante

```
try {
        // code pouvant lever une exception
        // son exécution ne se termine pas
        // si une exception est levée
} catch (TypeException1 e) {
        // gestion d'un type d'erreur
} catch (TypeException2 e) {
        // gestion d'un autre type d'erreur
// ... autant de catch que souhaité
} finally {
        // partie optionnelle toujours
        // exécutée ensuite
}
```

... ou en passant la patate chaude à la fonction appelante

```
... maMéthode(...) throws TypeException1 ...
```

Petit exemples

Java et les exceptions

Java gère deux grand types d'exceptions

- `Exception`, doivent être attrapées ou propagées explicitement
- `RuntimeException`, propagées automatiquement si non attrapées

Dans un bloc `try / catch / finally`

- une exception interrompt le bloc `try` qui ne se termine jamais
- les `catch` sont pris dans l'ordre jusqu'au premier type correspondant
- le `finally` (si présent) est toujours exécuté

Les exceptions se propagent en remontant dans la pile d'appels

- le programmeur peut attraper une exception au niveau qu'il souhaite
- permet de laisser remonter une erreur sans écrire de code particulier

Le bas niveau

`Scanner` et `PrintStream` travaillent sur des mots, lignes et entiers

- repésentés par plusieurs caractères consécutifs
- lus ou écrits caractère par caractère

On peut travailler à plus bas niveau, sur la représentation en machine des objets

- `DataInputStream` : méthodes `readByte`, `readShort`, ...
- `DataOutputStream` : méthodes `writeByte`, `writeShort`, ...

Un flux n'est plus vu comme un texte mais comme une séquence d'octets

Petit exemples

Performance

Pour les gros fichiers, les exemples précédents ne sont pas efficaces

- `InputStream` et `OutputStream` traduisent chaque lecture ou écriture par une requête au système
- on autrait intérêt à grouper les requêtes (lire 10, 100 ou 1000 caractères à l'avance par exemple)

On parle alors de mise en mémoire tampon (*Buffer*)

- on effectue des requêtes pour beaucoup plus d'octet que nécessaire
- on se sert dans le tampon tant qu'il n'est pas vide

En java, ce sera `BufferedInputStream` et `BufferedOutputStream`

Petit exemple