

JAVA SE - Objet Exception

Que transmettre dans un objet Exception ?

Tout programmeur Java sait qu'il a à définir et utiliser des Exceptions. Mais, souvent, la définition précise de l'objet Exception l'ennuie et ne donne pas lieu à une réflexion approfondie.

On voit alors facilement des codes comme celui-ci :

```
public class CompteEnBanque {  
    // divers champs  
    // dans le code d'un retrait  
    if( (this.solde - montantRetrait) < -this.decouvertAutorisé) {  
        throw new ExceptionRetraitCompte(  
            "Debit de " + montantRetrait + "impossible") ;  
    }  
}
```

Qu'est-ce qui ne va pas dans ce code? On a l'impression qu'une Exception a pour rôle principal de transmettre un message à l'utilisateur final! Or ce n'est pas du tout le cas.

Le rôle fondamental d'une Exception est de véhiculer un diagnostic applicatif. Ce diagnostic est émis par le code qui détecte l'anomalie à destination de code(s) appelant(s) qu'il ne connaît pas.

Certes un de ces codes appelant peut-être appelé à réagir et à prévenir le monde extérieur avec un message. Mais c'est au code qui capture l'Exception de générer un **rapport** qui aboutira à un diagnostic lisible par un être humain.

On aura donc un **rapport** (LogRecord dans le cas d'utilisation de la librairie java.util.logging) qui pourra être construit avec la chaîne "Debit de " + montantRetrait + "impossible" (ou une forme internationalisable de celle-ci). Ce rapport est certes construit à partir des informations contenues dans l'Exception mais le code émetteur de l'Exception n'a pas, d'une part, toutes les informations pour construire un rapport, et, d'autre part, n'a généralement pas pour rôle de proposer un message "utilisateur".

Le code du catch n'est pas forcément limité à signaler une anomalie à l'extérieur : il doit pouvoir analyser le diagnostic qu'il reçoit pour prendre des décisions d'ordre applicatif. Ce n'est certes pas l'analyse d'une chaîne de caractère qui va fournir le détail des informations nécessaires! Que ferait un tel code qui recevrait un diagnostic formulé ainsi : *"Suite à une anomalie réseau, nous ne sommes pas en mesure de vous fournir les 10 exemplaires des aventures de Bibi Fricotin que vous avez commandés"*!

Reprenons ces considérations du point de vue du code émetteur qui crée le diagnostic et déclenche sa propagation à d'autres codes.

Il est tout à fait possible que le code émetteur ne connaisse pas exactement les besoins des codes récepteurs (on est alors dans une architecture basée sur des composants). Dans ce cas il doit donc prévoir un éventail d'informations susceptibles d'être utiles (et utilisables!) : ces informations **sont des données**. Dans le cadre d'une application on doit concevoir, dans la phase de *design*, un protocole entre émetteurs et récepteurs.

Pour revenir à l'exemple précédent (commande de 10 livres) au lieu d'avoir un message verbeux inexploitable par un code applicatif on pourra décider de fournir dans l'Exception :

- Le livre manquant (normalement le code initiateur de l'action d'achat devrait le connaître, mais il est possible que d'autres codes aient à consulter les données de l'Exception)
- Le nombre d'exemplaires demandé par l'action d'achat (même remarque)
- L'état actuel du stock
- Une information permettant de sérier ensuite les incidents (marque horaire ou autre)
- Etc.

Quelques remarques :

- La classe de l'Exception peut définir un véritable objet (avec de nombreuses méthodes) ou un simple *data object* dont tous les champs sont public et final. On a, au fond, un diagnostic immuable qui sert à transporter des données d'une partie de l'application vers une autre.
- Le cas "Suite à une anomalie réseau" est à considérer à part mais on peut aussi avoir un chaînage d'exception. ExceptionAchat peut avoir une cause : c'est à dire faire référence à une autre Exception. (Dans ce cas il est possible d'avoir des éléments du diagnostic qui ne sont pas renseignés -comme l'état du stock-).
- L'objet qui déclenche l'Exception est souvent amené à passer sa propre référence au constructeur de l'Exception (par exemple le code de Livre déclenche ExceptionAchat et lui passe sa propre référence). On a dans ce cas un risque de couplage (Livre connaît ExceptionAchat et ExceptionAchat connaît Livre). On peut donc avoir recours à des solutions classiques concernant les couplages (utilisation d'une interface, définition de l'Exception comme classe interne, ...).
- Les données contenues dans l'Exception doivent fournir une image instantanée de l'incident. Supposons que le code de Livre dispose d'une méthode getStock() on ne peut pas s'appuyer sur cette méthode pour décortiquer l'incident : le nombre de livres en stock peut avoir évolué entre le déclenchement de l'incident et l'exploitation des données de l'Exception! (d'où ici une copie de l'état du stock au moment de la création du diagnostic).

Doit-on en conclure qu'on ne peut pas passer de "message" dans une Exception? Pas tout à fait..

Il est explicitement prévu dans la définition de la classe Exception d'avoir une chaîne constituant un message (voir également méthode getMessage()).

Deux cas principaux d'utilisation :

- Une précision apportée à un diagnostic (au fond une chaîne est aussi une donnée) :
- ```
throw new ExceptionRetraitCompte(this, this.solde, montantRetrait, "compte bloqué");
```
- Des Exceptions qui ne concernent pas la gestion applicative comme des RuntimeExceptions

```
public void retrait(double montantRetrait) {
 if(montantRetrait < 0) { // on pourrait aussi intercepter les retraits de 0
 throw new IllegalArgumentException("retrait négatif");
 }
 • Quoiqu'on aurait pu aussi faire évoluer ce code en définissant ceci :
public class ExceptionArgumentNegatif extends IllegalArgumentException {
 public final Number val ;
 public final String argument ;
 public ExceptionArgumentNegatif(Number val, String argument){
 this.val = val ;
 this.argument = argument ;
 }
 public String toString(){
 return super.toString()
 + " " + argument
 + " " + val ;
 }
}
```

Il existe, par ailleurs, quelques cas très particuliers dans lesquels les Exceptions sont effectivement chargés de véhiculer uniquement des messages pour l'utilisateur mais une description précise de ces rares cas dépasserait le périmètre de cette note.

## Formations JAVA sur ce sujet

Pour cadrer les notions fondamentales concernant les exceptions. Pour comprendre pourquoi et comment il faut distinguer les exceptions contrôlées des autres exceptions vous pouvez suivre les cours suivants :

- [USL110 : Java: bases du langage](#) (pour programmeurs débutants)
- [USL275: Java : Programmation pour Développeur Confirmé](#)
- [même programme mais sur 7 jours pour programmeurs débutants](#)
- Nos formations longues (standard ou spécifiques).