# Annexe 9 – Traitement des exceptions

Pour gérer les situations inhabituelles …

En procédural ( C ) :

* Alourdit le code ( plusieurs ifs )
* Le rend alors difficile à comprendre
* Découragement vis-à-vis cette tâche

En orienté-objet (Java ) :

* On traite les situations inhabituelles / problématiques selon le contexte à l’aide d’objets Exception

## Avantages de cette méthode

1. Séparation entre le traitement des exceptions et le déroulement normal du programme ( évite le code spaghetti, perd le fil conducteur )
2. Richesse des objets : peut contenir plusieurs infos complémentaires sur l’erreur en question plutôt qu’un simple code d’erreur ( valeur qui a causé l’erreur , etc )
3. Traitement de la situation problématique à l’endroit voulu seulement :

Activité ( interface graphique lecteur musique )

Classe ListeLecture

Méthode trierOrdreDuree

Classe Fichier

Variable nbSec <0 ???

\*\*\*On peut traiter l’exception à l’endroit désiré ( l’exception se propage dans la pile d’appel tant qu’elle n’est pas captée pour être traitée )

1. Hiérarchie des objets Exception : une erreur de lecture ( transmettre une string quand on demande un int ) peut être considérée comme telle ou comme une erreur générale d’entrée – sortie.

Hiérarchie des classes :

ORANGE = CONTROLLER

BLEU = NON CONTROLLER

AUTRE = ERREUR

Object

Throwable ( interface )

Error

Exception

SQLException IOException Nos propres classes RuntimeException StackOverflowError OutOfMemoryError

NullPointerException IndexOurOfBoundsException ArithmeticException NumberFormatException  **…**

**Erreurs : Ce sont des problèmes dont on ne peut pas récupérer de … donc on ne peu**

**Exceptions non-contrôlées ( unchecked ) : ce sont des sous classes de RuntimeException**

* **Survient lors de l’exécution du programme et non pas lors de sa compilation**
* NullPointerException : lorsqu’on appelle une méthode sur une référence null
* ArithmeticException : division par 0
* NumberFormatException : transtypage impossible entre « bonjour » ne se transtype pas en int ou en double

**Traitement des exceptions non-contrôlées**

Exercice Placement : Récupérer le dossier Placement présent sur LÉA. Dans un nouveau projet, intégrez la classe Placement, l’activité ainsi que le fichier de positionnement xml fournis sur LEA

1. Regarder la classe Placement
2. Regarder le fichier de positionnement XML, mieux répartir les composantes avec des poids
3. Gérer l’événement clic sur le bouton de manière à afficher le résultat de l’investissement
4. Qu’arrive-t-il si on écrit du texte dans le champ texte plutôt qu’un montant ?

* Solution try / catch

Voir projet Placement

## En résumé blocs try / catch

**Bloc try :**

🡪 On isole ds le bloc try la / les lignes qui peuvent lancer une exception

🡪 On porte attention aux lignes couplées qu’on place elles aussi ds le bloc try

try

{

1

2 🡪 NullPointerException est lancée : ni le résultat de la ligne 2, ni les lignes 3 et 4 ne sont exécutées, on est en mode exception ( balle ds les airs ) et on cherche un catch

3

4

}

**0/1 / plusieurs blocs catch associés**

🡪 si l’exception correspond au catch, les lignes de code du bloc catch sont exécutées

🡪 le compilateur ne revient pas ds le try une fois l’exception captée, il continue à la suite du bloc catch

\*\*\*les blocs catch doivent être organisés du plus spécifique vers le plus général sinon des blocs deviendraient inutiles car jamais atteints 🡪 une exception ne peut être captée qu’une fois

* Finalement si le ou les blocs catch ne correspondent pas à l’exception lancée, elle remonte la pile d’appel à la recherche d’un catch qui saura l’accueillir…tout comme si une exception est lancée à l’extérieur d’un bloc try

Structure try / catch : commune dans plusieurs langages

## Autre bloc possible : finally

* Facultatif, qu’on place à la suite du catch, utile pour placer des méthodes qui doivent absolument être exécutées ( fermer connexion sql, lien vers un fichier texte )

**IMPORTANT**

**On bloc Finally est toujours execute, qu’une exception ait été lancée précédemment dans le bloc Try ou non.**

try

{

}

catch ( NullPointerException npe )

{

}

finally

{

}

\*\*\*\* Si pas d’exception : on fait le try + finally

\*\*\* Si NullpointerException est lancee : Try jusqu’au probleme + catch + finally

\*\*\* Si un autre type d’exception est lance : try jusqu’au probleme + finally + on recherche un catch plus haut dans la pile d’appel.