Les exceptions

Procédure partielle :
Clause @requires présente -> certains inputs ne sont pas gérés

Procédure totale (pour une meilleure robustesse par ex.) :
Pas de @requires -> tous les inputs sont gérés

Procédure partielle -> procédure totale : on pourrait par ex. Retourner -1 pour la recherche d'un indice de tableau, ou retourner un NotSuchElementException

- Mettre en avant un cas particulier
- Pas de retour disponible pour un cas particulier
- Simplification de code (abstraction et implémentation)

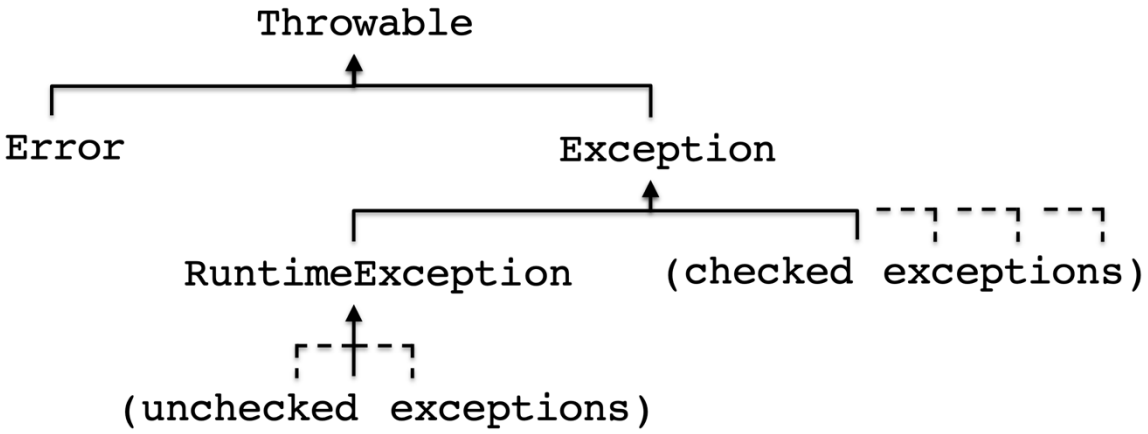
Une exception n'est pas un erreur. Elle peut, par exemple, être attendue en tant que valeur possible/particulière de réponse.

Type d'exception

L'exception peut être vérifiée ou non vérifiée :

Vérifiée	**Non vérifiée**
extends Exception Contrainte de compilation de Java (le compilateur vérifie) : - Pour la procédure qui peut lever l'exception, l'Exception doit être dans la signature. Ex: public int foo() throws Exception; - Au niveau du code appelant, s'il n'y a pas de traitement de l'exception, il y a une erreur de compilation Au niveau des spécifiactions il faut spécifier les exceptions via des classes **@throws** et indiquer le comportement au niveau des @effects et @return. La clause **@throws** indique l'exception jetée et sa cause Ex : @throws NullPointeurException si a est null/non défini	extends RuntimeException Pas de contrainte de compilation

Toutes les exceptions étendent Throwable.



Création de nouvelles exceptions

public class MonException extends Exception pour une exception vérifiée
public class MaNonVerifieException extends RuntimeException pour une exception non vérifiée

Un template conseillé/imposé :

```
/** Exception vérifiée qui … */
public class MonException extends Exception {

        public MonException(){
                super();
        }

        Public MonException(String s){
                super(s);
        }
}
```

Jeter l'exception : mot clef **throw**

Ex : throw new MonException("blabla");

Gestion des exceptions

1ère manière : capture (gestion interne au code appelant)

```
try{
        // Appel d'une méthode qui peut jeter une exception
}
catch (MonException e){

}
catch (AException e1 | BException e2){

}
finally{
        // toujours exécuté !!!
        // pas sensible au break, return, exit…
}
```

2ème manière : propagation

On utilise pas le try catch, mais on ajoute l'exception dans la signature de la méthode.
Ainsi on délègue la gestion de l'exception au code appelant.
L'erreur peut ainsi se propager et remontrer le programme jusqu'à gestion ou … crash de l'application.

Bonnes pratiques

- Le nom d'une exception se termine par Exception; ex: valeurAbsenteException
- Au niveau de la gestion "interne" d'une exception, être le plus précis possible au niveau de la capture
- Message explicite qui contient idéalement la cause et contient TOUJOURS l'origine de l'exception
- Positionnement : un package commun/indépendant à toutes les exceptions du projet; si impossible au même niveau que le code appelant
- Traiter les exceptions non vérifiées comme des exceptions vérifiées (sauf FailureException)

FailureException

Utilisée quand il y a un problème de programmation ou un problème qui ne peut être géré/prévu par le développeur (pb de ressource, erreur matérielle,…)

Spécification

Les clauses @throws se trouvent soit après la clause @modifies, soit après les clauses @effects ou @return. Les spécifications seront adaptées en fonction.

```
/**
* @requires v != null
```

```
* @modifies v
* @throws NullPointerException si x est null
* @throws NotSmallException si v contient un´el´ement > x

* @effects sinon, ajoute x dans v
*/
public static void addMax(List<Integer> v, Integer x) throws NullPointerException,
NotSmallException {

... }

/**
* @requires v != null
* @modifies v
* @effects ajoute x dans v,
* @throws NullPointerException si x est null
* @throws NotSmallException si v contient un´el´ement > x

*/
public static void addMax(List<Integer> v, Integer x) throws NullPointerException,
NotSmallException {

... }
```