Les exceptions

Introduction

- A la compilation, le compilateur cherche à éliminer le plus d ’erreurs possible, en particulier les erreurs de syntaxe
- Des erreurs peuvent néanmoins survenir lors de l ’exécution du programme :
  - elles dépendent du « contexte » d ’exécution (état des variables, accessibilité des périphériques, etc.)
  - elles ne sont pas identifiées par le compilateur (en amont du processus d ’exécution)
  - tout bon programmeur doit envisager et gérer les cas (valeurs des variables incohérentes, etc.) qui sont susceptibles de créer une erreur d ’exécution

Principe

- Les exceptions sont un moyen pratique, fourni par Java, de faire le contrôle d'erreurs pouvant surgir lors de l'exécution.
- Une exception est un signal qui indique que quelque chose d'exceptionnel est survenu en cours d'exécution
- Deux solutions alors :
  - laisser le programme se terminer avec une erreur
  - essayer, malgré l'exception, de continuer l'exécution normale
- Lancer une exception consiste à signaler quelque chose d'exceptionnel
- Capturer l'exception consiste à signaler qu'on va la traiter

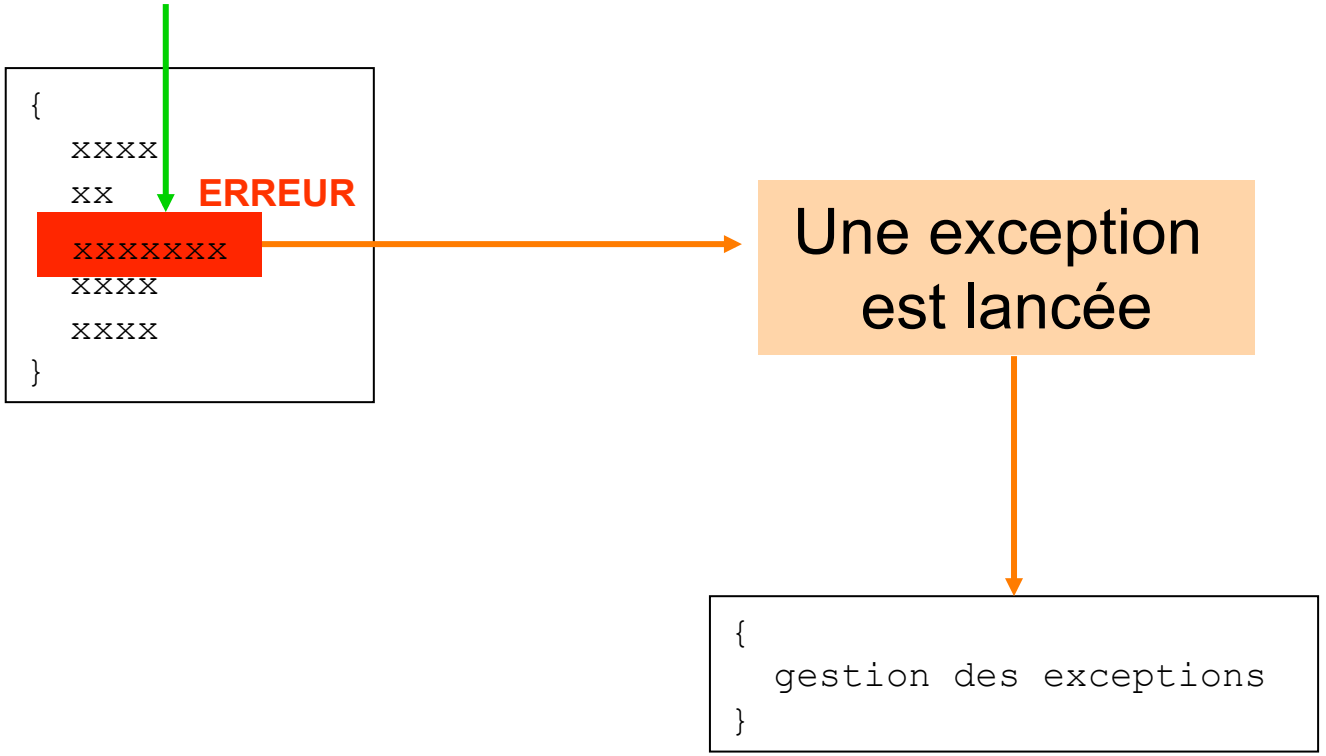
Quelques exceptions préexistant dans Java

- Division par zéro pour les entiers : ArithmeticException
- Déréférencement d'une référence nulle : NullPointerException
- Tentative de forçage de type illégale : ClassCastException
- Tentative de création d'un tableau de taille négative : NegativeArraySizeException
- Dépassement de limite d'un tableau : ArrayIndexOutOfBoundsException

Des exceptions pour écrire du code fiable

- Java exige qu'une méthode susceptible de lancer une exception (hormis les Error et les RuntimeException) indique quelle doit être l'action à réaliser.
  - Sinon, il y a erreur de compilation.
- Le programmeur a le choix entre :
  - écrire un bloc try / catch pour traiter l'exception,
  - laisser remonter l'exception au bloc appelant grâce à un throws.
- C'est ce qu'on appelle : "Déclarer ou traiter".

Capter les exceptions



Capter les exceptions

- Pour capter une exception, il faut placer les instructions à surveiller dans un bloc `try`
- Syntaxe :

```
try
{
  bloc d'instructions
}
catch(exception-type identifier)
{
  bloc d'instructions (gestion de l'erreur)
}
catch(exception-type identifier)
{
  bloc d'instructions (gestion de l'erreur)
}
finally
{
  bloc d'instructions
}
```

Capter les exceptions

- Le bloc `try` est exécuté jusqu'à ce qu'il se termine avec succès ou bien qu'une exception soit levée.
- Dans ce dernier cas, les clauses `catch` sont examinées l'une après l'autre dans le but d'en trouver une qui traite cette classe d'exceptions (ou une superclasse).
- Les clauses `catch` doivent donc traiter les exceptions de la plus spécifique à la plus générale.
  - La présence d'une clause `catch` qui intercepte une classe d'exceptions avant une clause qui intercepte une sous-classe d'exceptions déclenche une erreur de compilation.
- Si une clause `catch` convenant à cette exception a été trouvée et le bloc exécuté, l'exécution du programme reprend son cours.

Capter les exceptions

- Si elles ne sont pas immédiatement capturées par un bloc `catch`, les exceptions se propagent en remontant la pile d'appels des méthodes, jusqu'à être traitées.
- Si une exception n'est jamais capturée, elle se propage jusqu'à la méthode `main()`, ce qui pousse l'interpréteur Java à afficher un message d'erreur et à s'arrêter.
- L'interpréteur Java affiche un message identifiant :
  - l'exception,
  - la méthode qui l'a causée,
  - la ligne correspondante dans le fichier.

Capter les exceptions

- Exemple :

```
try
{
  instructions…
}
catch(java.io.IOException ioe)
{
  System.err.println(« Probleme d ’e/s : « +ioe);
}
catch(Exception e)
{
  System.err.println(« Probleme : « +e);
}
```

La clause finally

- L'écriture d'un bloc `finally` permet au programmeur de définir un ensemble d'instructions qui est toujours exécuté, que l'exception soit lancée ou non, capturée ou non.
- Si une clause `finally` est présente dans une instruction `try`, son code est exécuté après les instructions contenues dans le bloc try, quelle que soit la façon dont le traitement s'est achevé, que ce soit normalement, par une exception, ou par une instruction de contrôle comme return ou break.
- La tâche habituelle assignée à une clause `finally` est de réinitialiser l'état interne de la mémoire, ou de libérer les ressources qui ne sont pas des objets comme, par exemple, les fichiers ouverts qui sont stockés dans des variables locales

throws

- Pour "laisser remonter" à la méthode appelante une exception qu'il ne veut pas traiter, le programmeur ajoute le mot réservé `throws` à la déclaration de la méthode dans laquelle l'exception est susceptible de se manifester.

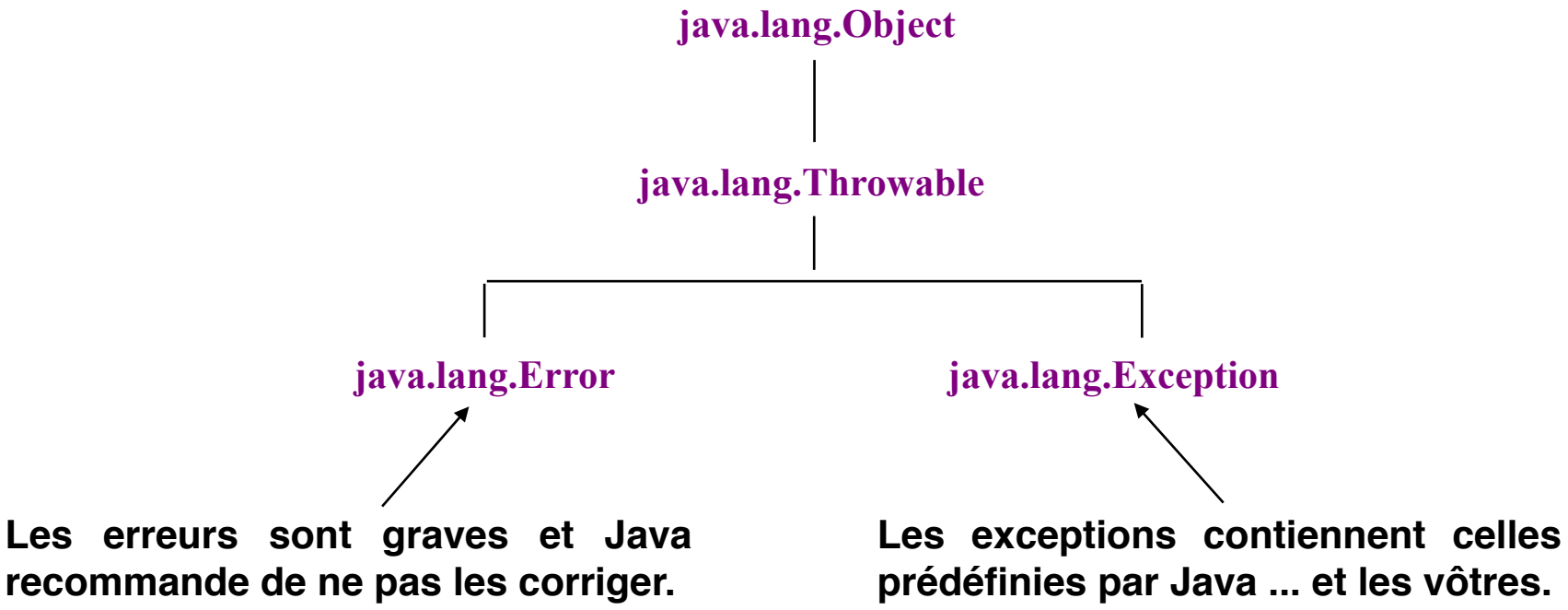
```
public void uneMethode() throws IOException
    {
        // ne traite pas l'exception IOException
        // mais est susceptible de la générer
    }
```

throws

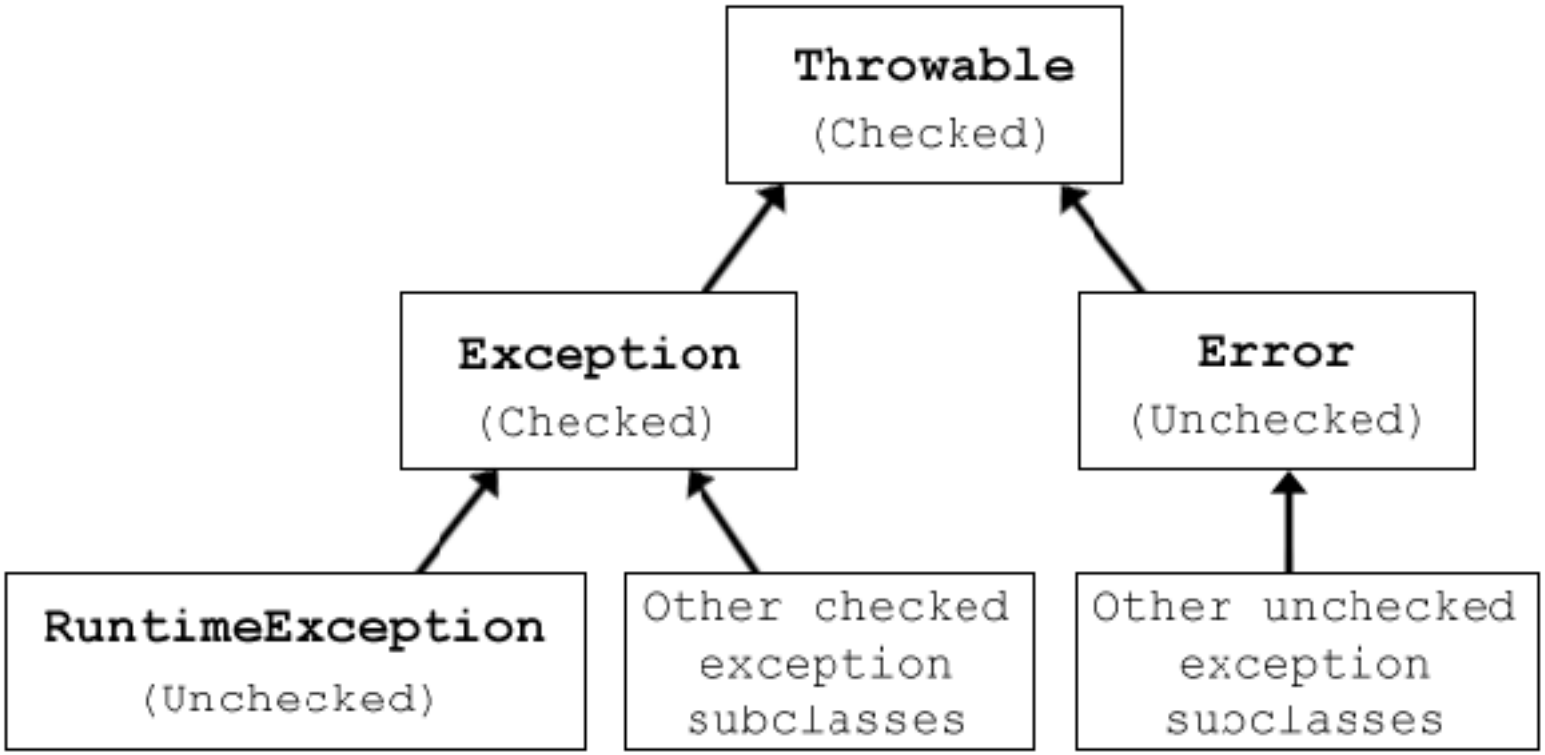
- Les programmeurs qui utilisent une méthode connaissent ainsi les exceptions qu'elle peut lever.
- La classe de l'exception indiquée peut tout à fait être une super-classe de l'exception effectivement générée.
- Une même méthode peut tout à fait "laisser remonter" plusieurs types d'exceptions (séparés par des `,`).
- Une méthode doit traiter ou "laisser remonter" toutes les exceptions qui peuvent être générées dans les méthodes qu'elle appelle (et ceci récursivement).

Les objets `Exception`

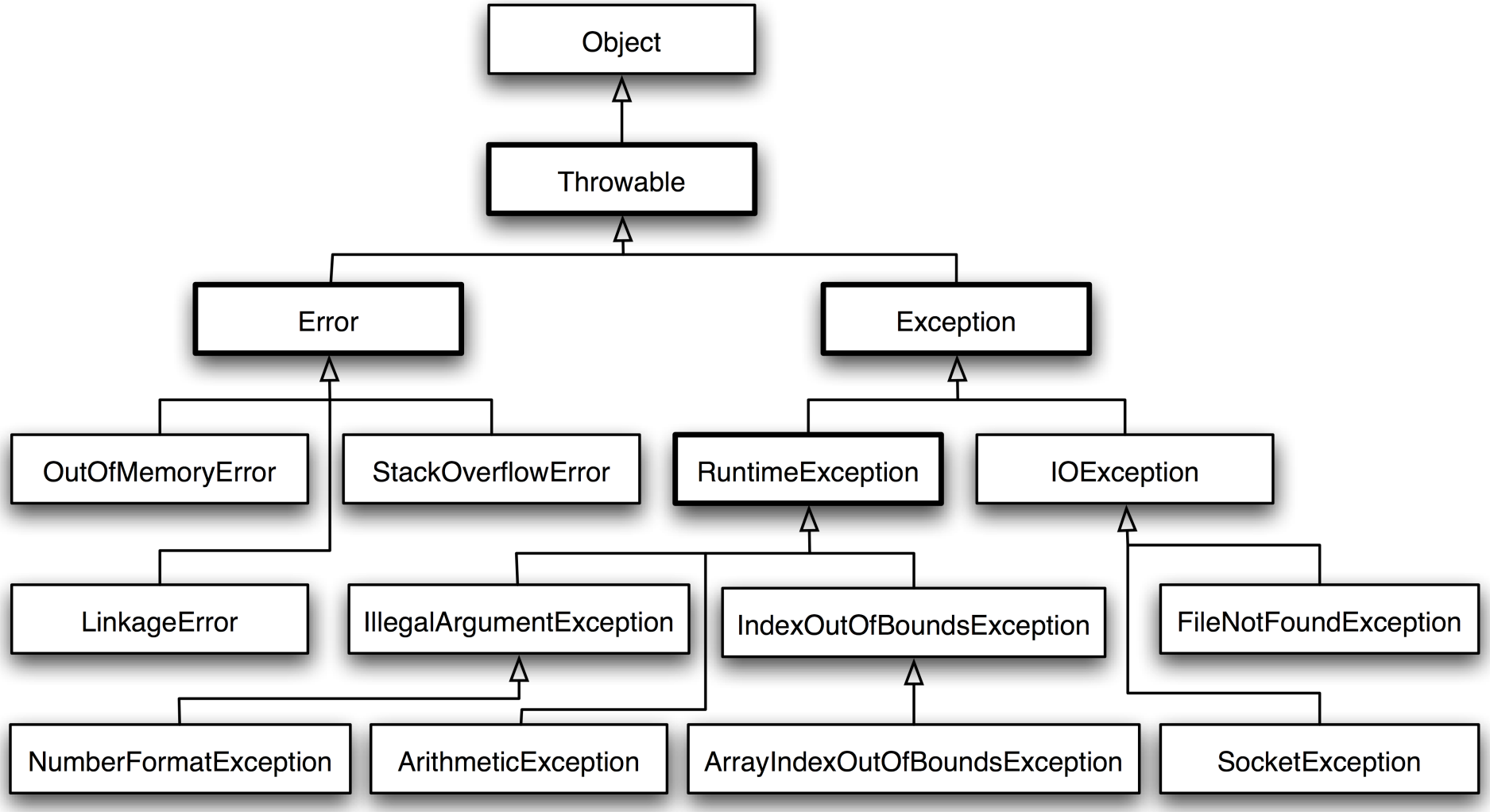
- Dans Java, il existe deux types d'objets Throwable, représentées par deux classes de l'API Java.



Les objets Exception



Les objets `Exception`



Les objets `Exception`

- La classe `Throwable` définit un message de type `String` qui est hérité par toutes les classes d'exception.
- Ce champ est utilisé pour stocker le message décrivant l'exception.
- Il est positionné en passant un argument au constructeur.
- Ce message peut être récupéré par la méthode `getMessage()`.

Les objets `Exception`

- Exemple :

```
public class MonException extends Exception
{
    public MonException()
    {
      super();
    }
    public MonException(String s)
    {
      super(s);
    }
}
```

Levée d'exceptions

- Le programmeur peut lancer ses propres exceptions à l'aide du mot réservé `throw`.
- `throw` prend en paramètre un objet instance de `Throwable` ou d'une de ses sous-classes.
- Les objets exception sont souvent alloués dans l'instruction même qui assure leur lancement.

```
throw new MonException("Mon exception s'est produite !!!");
```