# Cours java : Les exceptions

Emmanuel Coquery
emmanuel.coquery@liris.cnrs.fr

## 1 Introduction : qu'est-ce qu'une exception ?

De nombreux langages de programmation de haut niveau possèdent un mécanisme permettant de gérer les erreurs qui peuvent intervenir lors de l'exécution d'un programme. Le mécanisme de gestion d'erreur le plus répandu est celui des exceptions. Nous avons déjà abordé le concept d'exception dans le cours sur les fonctions : lorsqu'une fonction n'est pas définie pour certaines valeur de ses arguments on lance une exception. Par exemple, la fonction factorielle n'est pas définie pour les nombres négatifs :

```
class Factorielle {
    static int factorielle(int n){
        int res = 1;
        if (n<0){
            throw new PasDefini();
        }
        for(int i = 1; i <= n; i++) {
            res = res * i;
        }
        return res;
    }
}

class PasDefini extends Error {}
```

Une exception représente une erreur. Il est possible de lancer une exception pour signaler une erreur, comme lorsqu'un nombre négatif est passé en argument à la fonction factorielle. Jusqu'ici, lever une exception signifiait interrompre définitivement le programme avec un message d'erreur décrivant l'exception en question. Cependant, il est de nombreuses situations où le programmeur aimerait gérer les erreurs sans que le programme ne s'arrête définitivement. Il est alors important de pouvoir intervenir dans le cas où une exception a été levée. Les langages qui utilisent les exceptions possèdent toujours une construction syntaxique permettant de "rattraper" une exception, et d'exécuter un morceau de code avant de reprendre l'exécution normale du programme.

En Java, les exceptions sont des objets représentant les erreurs qui peuvent survenir au cours de l'exécution d'un programme.

## 2 Définir des exceptions

Afin de définir une nouvelle sorte d'exception, on crée une nouvelle classe en utilisant une déclaration de la forme suivante :

```
class NouvelleException extends ExceptionDejaDefinie {}
```

On peut remarquer ici la présence du mot clé **extends**, dont nous verrons la signification dans un chapitre ultérieur qui traitera de l'héritage entre classes. `NouvelleException` est le nom de la classe d'exception que l'on désire définir en "étendant" `ExceptionDejaDefinie` qui est une classe d'exception déjà définie. Sachant que `Error` est prédéfinie en Java, la déclaration suivante définit bien une nouvelle classe d'exception `PasDefini` :

```
class PasDefini extends Error {}
```

Il existe de nombreuses classes d'exceptions prédéfinies en Java, que l'on peut classer en trois catégories :

- Celles définies en étendant la classe `Error` : elles représentent des erreurs critiques qui ne sont pas censée être gérées en temps normal. Par exemple, une exception de type `OutOfMemoryError` est lancée lorsqu'il n'y a plus de mémoire disponible dans le système.
- Celles définies en étendant la classe `Exception` : elles représentent les erreurs qui doivent normalement être gérées par le programme. Par exemple, une exception de type `IOException` est lancée en cas d'erreur lors de la lecture d'un fichier.
- Celles définies en étendant la classe `RuntimeException` : elles représente des erreurs pouvant éventuellement être gérée par le programme. L'exemple typique de ce genre d'exception est `NullPointerException`, qui est lancée si l'on tente d'accéder au contenu d'un tableau ou d'un objet qui vaut `null`.

Chaque nouvelle exception définie entre ainsi dans l'une de ces trois catégories. Si on suit ce classement, l'exception `PasDefini` aurait due être déclarée par :

```
class PasDefini extends Exception {}
```

ou bien par :

```
class PasDefini extends RuntimeException {}
```

car elle ne constitue pas une erreur critique.

# 3 Lancer une exception

Lorsque l'on veut lancer une exception, on utilise le mot clé `throw` suivi de l'exception à lancer, qu'il faut auparavant la créer avec `new NomException()`, de la même manière que lorsque l'on crée un nouvel objet par un appel à l'un de des constructeurs de sa classe. Ainsi lancer une exception de la classe `PasDefini` s'écrit :

```
throw  new PasDefini();
```

Lorsqu'une exception est levée, l'exécution normale du programme s'arrête et on saute toutes les instructions jusqu'à ce que l'exception soit rattrapée ou jusqu'à ce que l'on sorte du programme. Par exemple, si on considère le code suivant [1] :

```
public class Arret {
    public static void main(String [] args) {
        Terminal.ecrireStringln("Coucou 1");       // 1
        if (true) throw new Arret();
        Terminal.ecrireStringln("Coucou 2");       // 2
        Terminal.ecrireStringln("Coucou 3");       // 3
        Terminal.ecrireStringln("Coucou 4");       // 4
    }
}
class Stop extends RuntimeException {}
```

alors l'exécution de la commande `java Arret` produira l'affichage suivant :

```
> java Arret
Coucou 1
Exception in thread "main" Stop
        at Arret.main(Arret.java:5)
```

C'est-à-dire que les instructions 2, 3 et 4 n'ont pas été exécutées. Le programme se termine en indiquant que l'exception `Stop` lancée dans la méthode `main` à la ligne 5 du fichier `Arret.java` n'a pas été rattrapée.

1. Dans ce code, on a ajouté `if (true)` avant le `throw new Arret()` afin de tromper le compilateur. Sans cela la compilation produit une erreur en expliquant que le code 2 ne sera jamais exécuté.

# 4 Rattraper une exception

## 4.1 La construction `try catch`

Le rattrapage d'une exception en Java se fait en utilisant la construction :

```
try {
   ... // 1
} catch (UneException e) {
   ... // 2
}
.. // 3
```

Le code 1 est normalement exécuté. Si une exception est lancée lors de cette exécution, les instructions restantes dans le code 1 sont sautées. Si la classe de l'exception est `UneException` alors le code 2 est exécuté. Dans le code 2, on peut faire référence à l'exception en utilisant le nom donné à celle-ci après sa classe (ici le nom est `e`). Si la classe de l'exception n'est pas `UneException`, le code 2 et le code 3 sont sautés. Ainsi, le programme suivant :

```
public class Arret {
    public static void main(String [] args) {
try {
    Terminal.ecrireStringln("Coucou 1"); // 1
    if (true) throw new Stop();
    Terminal.ecrireStringln("Coucou 2"); // 2
} catch (Stop e) {
    Terminal.ecrireStringln("Coucou 3"); // 3
}
Terminal.ecrireStringln("Coucou 4"); // 4
    }
}

class Stop extends RuntimeException {}
class Stop2 extends RuntimeException {}
```

produit l'affichage suivant lorsqu'il est exécuté :

```
> java Arret
Coucou 1
Coucou 3
Coucou 4
```

En revanche le programme suivant, dans lequel on lance l'exception `Stop2`,

```
public class Arret {
    public static void main(String [] args) {
try {
    Terminal.ecrireStringln("Coucou 1"); // 1
    if (true) throw new Stop2();
    Terminal.ecrireStringln("Coucou 2"); // 2
} catch (Stop e) {
    Terminal.ecrireStringln("Coucou 3"); // 3
}
Terminal.ecrireStringln("Coucou 4"); // 4
    }
}
```

```
class Stop extends RuntimeException {}
class Stop2 extends RuntimeException {}
```

produit l'affichage suivant lorsqu'il est exécuté :

```
> java Arret
Coucou 1
Exception in thread "main" Stop2
        at Arret.main(Arret.java:5)
```

## 4.2 Rattraper plusieurs exceptions

Il est possible de rattraper plusieurs types d'exceptions en enchaînant les constructions `catch` :

```
public class Arret {
    public static void main(String [] args) {
try {
    Terminal.ecrireStringln("Coucou 1"); // 1
    if (true) throw new Stop2();
    Terminal.ecrireStringln("Coucou 2"); // 2
} catch (Stop e) {
    Terminal.ecrireStringln("Coucou 3"); // 3
} catch (Stop2 e) {
    Terminal.ecrireStringln("Coucou 3 bis"); // 3
}
Terminal.ecrireStringln("Coucou 4"); // 4
    }
}

class Stop extends RuntimeException {}
class Stop2 extends RuntimeException {}
```

A l'exécution, on obtient :

```
>   java Arret
Coucou 1
Coucou 3 bis
Coucou 4
```

Il est également possible d'imbriquer les constructions `try catch` :

```
public class Arret {
    public static void main(String [] args) {
try {
    try {
Terminal.ecrireStringln("Coucou 1"); // 1
if (true) throw new Stop();
Terminal.ecrireStringln("Coucou 2"); // 2
    } catch (Stop e) {
Terminal.ecrireStringln("Coucou 3"); // 3
    }
    Terminal.ecrireStringln("Coucou 4"); // 4
} catch (Stop2 e) {
    Terminal.ecrireStringln("Coucou 5"); // 5
}
Terminal.ecrireStringln("Coucou 6"); // 6
    }
```

```
}

class Stop extends RuntimeException {}
class Stop2 extends RuntimeException {}
```

L'exécution produit l'affichage suivant :

```
> java Arret
Coucou 1
Coucou 3
Coucou 4
Coucou 6
```

En remplaçant `throw new Stop()` par `throw new Stop2()`, on obtient :

```
> java Arret
Coucou 1
Coucou 5
Coucou 6
```

# 5 Exceptions et méthodes

## 5.1 Exception non rattrapée dans le corps d'une méthode

Si une exception lancée lors de l'exécution d'une méthode n'est pas rattrapée, elle "continue son trajet" à partir de l'appel de la méthode. Même si la méthode est sensée renvoyer une valeur, elle ne le fait pas :

```
public class Arret {
    static int lance(int x) {
if (x < 0) {
    throw new Stop();
}
return x;
    }

    public static void main(String [] args) {
int y = 0;
try {
    Terminal.ecrireStringln("Coucou 1");
    y = lance(-2);
    Terminal.ecrireStringln("Coucou 2");
} catch (Stop e) {
    Terminal.ecrireStringln("Coucou 3");
}
Terminal.ecrireStringln("y vaut "+y);
    }
}

class Stop extends RuntimeException {}
```

A l'exécution on obtient :

```
> java Arret
Coucou 1
Coucou 3
y vaut 0
```

### 5.2 Déclaration `throws`

Le langage Java, demande à ce que soient déclarées les exceptions et qui peuvent être lancées dans une méthode sans être rattrapées dans cette même méthode. Cette déclaration est de la forme `throws Exception1, Exception2, ...` et se place entre les arguments de la méthode et l'accolade ouvrant marquant le début du corps de la méthode. Cette déclaration n'est pas obligatoire pour les exceptions de la catégorie `Error` ni pour celles de la catégorie `RuntimeException`.

## 6 Exemple résumé

On reprend l'exemple de la fonction factorielle :

```
class Factorielle {
    static int factorielle(int n) throws PasDefini { // (1)
        int res = 1;
        if (n<0){
            throw new PasDefini();                   // (2)
        }
        for(int i = 1; i <= n; i++) {
            res = res * i;
        }
        return res;
    }

    public static void main (String [] args) {
        int x;
        Terminal.ecrireString("Entrez un nombre (petit):");
        x = Terminal.lireInt();
        try {                                        // (3)
            Terminal.ecrireIntln(factorielle(x));
        } catch (PasDefini e) {                      // (3 bis)
            Terminal.ecrireStringln("La factorielle de "
                                    +x+" n'est pas définie !");
        }
    }
}

class PasDefini extends Exception {}                 // (4)
```

Dans ce programme, on définit une nouvelle classe d'exception `PasDefini` au point (4). Cette exception est lancée par l'instruction `throw` au point (2) lorsque l'argument de la méthode est négatif. Dans ce cas l'exception n'est pas rattrapée dans le corps et comme elle n'est ni dans la catégorie `Error` ni dans la catégorie `RuntimeException`, on la déclare comme pouvant être lancée par factorielle, en utilisant la déclaration `throws` au point (1). Si l'exception `PasDefini` est lancée lors de l'appel à factorielle, elle est rattrapée au niveau de la construction `try catch` des points (3) (3 bis) et un message indiquant que la factorielle du nombre entré n'est pas définie est alors affiché. Voici deux exécutions du programme avec des valeurs différentes pour `x` (l'exception est lancée puis rattrapée lors de la deuxième exécution) :

```
> java Factorielle
Entrez un nombre (petit):4
24
> java Factorielle
Entrez un nombre (petit):-3
La factorielle de -3 n'est pas définie !
```

## Annexe : quelques exceptions prédéfinies

Voici quelques exceptions prédéfinies dans Java :

- `NullPointerException` : accès à un champ ou appel de méthode non statique sur un objet valant `null`. Utilisation de `length` ou accès à un case d'un tableau valant `null`.
- `ArrayIndexOutOfBoundsException` : accès à une case inexistante dans un tableau.
- `ArrayIndexOutOfBoundsException` : accès au $i^{eme}$ caractère d'un chaîne de caractères de taille inférieure à $i$.
- `ArrayIndexOutOfBoundsException` : création d'un tableau de taille négative.
- `NumberFormatException` : erreur lors de la conversion d'un chaîne de caractères en nombre.

La classe `Terminal` utilise également l'exception `TerminalException` pour signaler des erreurs.