



Java Avancé

Partie 3

Les exceptions

F. Belabdelli



Les exceptions

Définition

Hiérarchie des exceptions

Exemples d'exceptions

Interceptor (capturer) une exception

L'objet Exception

Remonte d'exception

Interceptor plusieurs exceptions

Déclencher une exception (throw)

Propager une exception (throws)

Définir sa propre exception

Bloc finally

Le try-with-resources

Un bloc catch, plusieurs exceptions



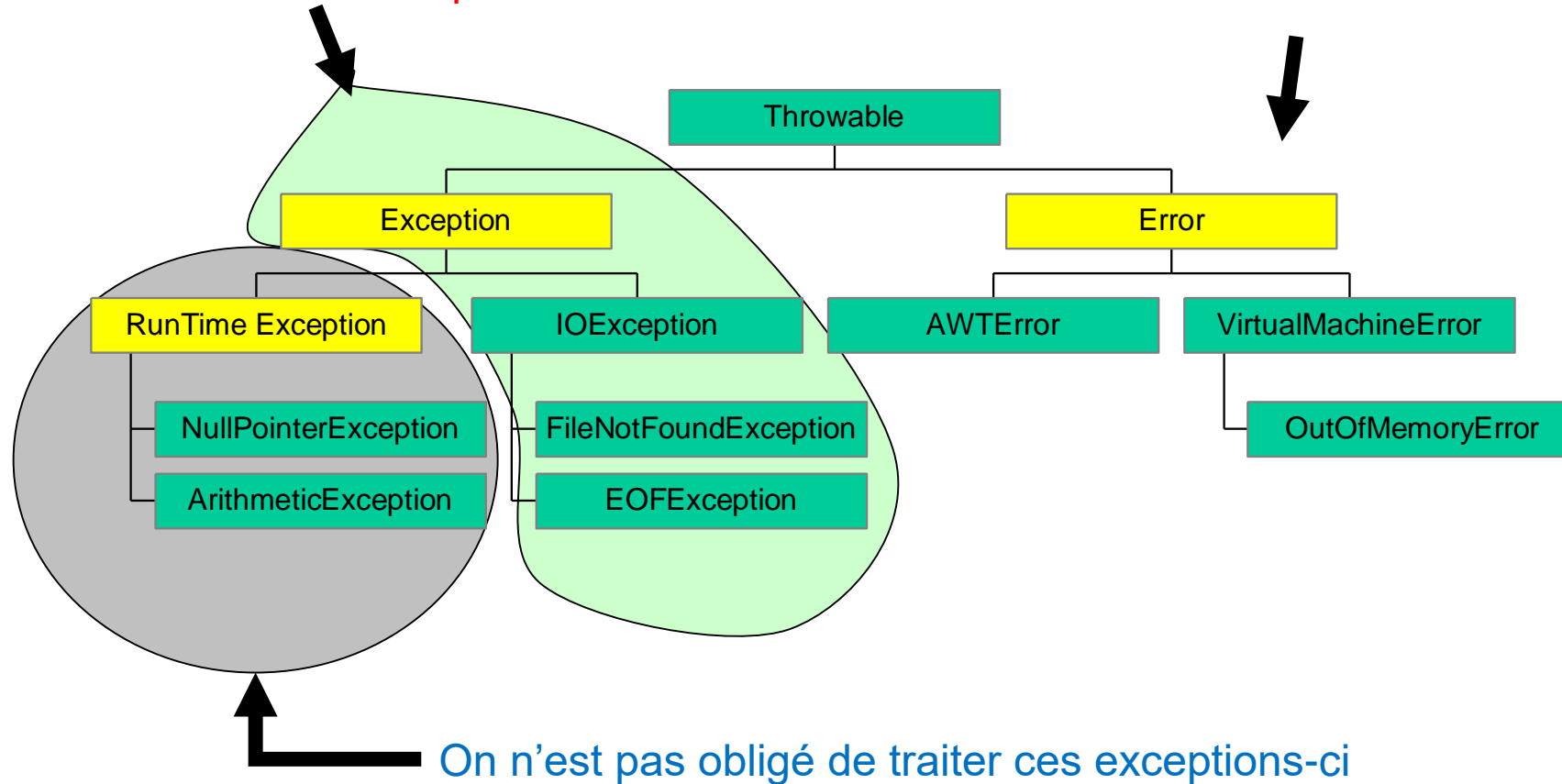
- Un programme peut être confronté à une condition exceptionnelle (ou exception) durant son exécution.
- Une exception est une situation qui empêche l'exécution normale du programme (elle ne doit pas toujours être considérée comme un bug).
- Le mécanisme de **gestion des exceptions** intégré au langage Java permet :
 - de séparer la détection d'une anomalie de son traitement
 - de séparer la gestion des anomalies du reste du code (meilleure lisibilité)
- Une exception déclenchée provoque un arrêt immédiat des traitements en cours et un branchement vers le gestionnaire d'exception.

Hiérarchie (nature) des exceptions



On doit traiter ces exceptions-ci

On ne peut pas traiter les « Error »



Exemples d'exceptions :



Débordement de l'indice du tableau

```
public static void main(String[] args) {  
    int [] tab = {1,5,8,10};  
    tab[6]=13;  
}
```

```
Console  
<terminated> TPTest (1) [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 6 out  
of bounds for length 4  
    at TPTest.main(TPTest.java:8)
```

Utilisation d'une référence nulle

```
public static void main(String[] args) {  
    ArrayList<String> liste=null;  
    for(String s:liste) {  
        System.out.println(s);  
    }  
}
```

```
Console  
<terminated> TPTest (1) [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.util.ArrayList.iterator()" because "liste" is null  
    at TPTest.main(TPTest.java:7)
```

Division par zéro

```
public static void main(String[] args) {  
    int a = 5, b=0 ;  
    int c = a/b ;  
}
```

```
Console  
<terminated> TPTest (1) [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at TPTest.main(TPTest.java:7)
```

Argument non conforme

```
public static void main(String[] args) {  
    String nombre="10.5";  
    int val = Integer.parseInt(nombre);  
}
```

```
Console  
<terminated> TPTest (1) [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (21 oct. 2021 à 15:43:43 -  
Exception in thread "main" java.lang.NumberFormatException: For input string: "10.5"  
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.base/java.lang.Integer.parseInt(Integer.java:652)  
    at java.base/java.lang.Integer.parseInt(Integer.java:770)  
    at TPTest.main(TPTest.java:6)
```

Exemples d'exceptions



- **ArithmeticException** : opération arithmétique impossible comme la division par 0.
- **IndexOutOfBoundsException** : dépassement d'indice dans un tableau, un vecteur, . . .
- **NullPointerException** : accès à un attribut/méthodes/case pour les tableaux d'une référence valant null, argument null alors que ce n'est pas autorisé.
- **NumberFormatException** : échec d'une conversion vers le type demandé
- **IllegalArgumentException** : argument incorrect (en dehors des valeurs autorisées) lors de l'appel d'une méthode.
- **NoSuchElementException** : next alors que l'itération est finie, dépilement d'une pile vide, . . .
- . . .

Interceptor (capturer) une exception



- Il faut placer les lignes de code pouvant produire une exception dans un bloc particulier nommé **try**.
- Il faut faire suivre ce bloc **d'un ou plusieurs** gestionnaires d'exceptions catch.
- Le gestionnaire peut être spécifique, il n'intercepte qu'un seul type d'exception. Il permet d'effectuer un traitement plus fin des problèmes selon les types d'exceptions.

```
public static void main(String[] args) {  
    String nombre="10.5";  
    {  
        try{  
            int val = Integer.parseInt(nombre);  
            System.out.println("val="+val);  
        }  
        {  
            catch(NumberFormatException e) {  
                System.out.println("Attention l'argument passé  
                n'est pas un entier !!");  
            }  
        }  
    }  
}
```

try { }

Bloc pouvant
produire une erreur

catch (...) { }

S'exécute en cas
d'erreur dans le bloc
try

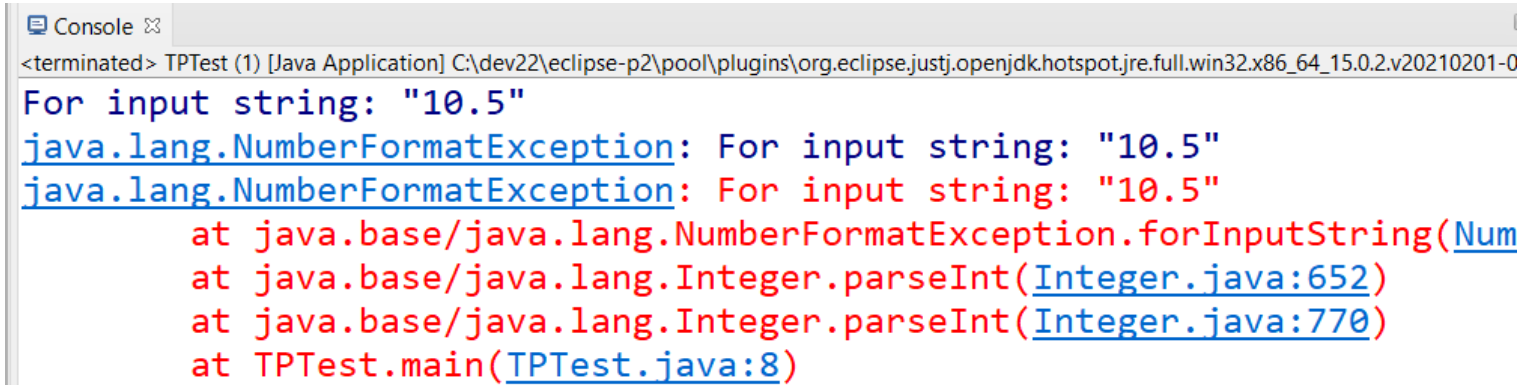
```
Console ✖  
<terminated> TPTest (1) [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre  
Attention l'argument passé n'est pas un entier !!
```

L'objet Exception :



- Le gestionnaire d'exception reçoit un objet lorsque l'exception est déclenchée.
- Cet objet possède les méthodes dont voici quelques unes :

```
public static void main(String[] args) {  
    String nombre="10.5";  
    try{  
        int val = Integer.parseInt(nombre);  
        System.out.println("val="+val);  
    }  
    catch(NumberFormatException e) {  
        System.out.println(e.getMessage());  
        System.out.println(e.toString());  
        e.printStackTrace();  
    }  
}
```



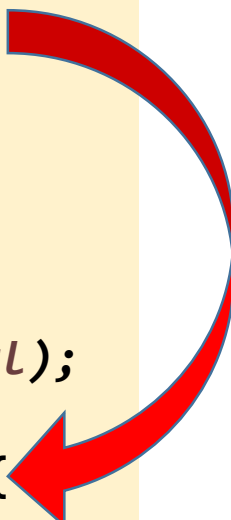
```
Console  
<terminated> TPTest (1) [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0  
For input string: "10.5"  
java.lang.NumberFormatException: For input string: "10.5"  
java.lang.NumberFormatException: For input string: "10.5"  
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.base/java.lang.Integer.parseInt(Integer.java:652)  
    at java.base/java.lang.Integer.parseInt(Integer.java:770)  
    at TPTest.main(TPTest.java:8)
```


Remonte d'exception vers la méthode appelante



- Dans le cas où la méthode ne gère pas l'erreur, le gestionnaire d'exception remonte l'objet exception à la méthode appelante.

```
public int convertir(String s) {  
    int val = Integer.parseInt(s);  
    return val;  
}  
public void traiterNombre() {  
    try{  
        int val = convertir("10.5");  
        System.out.println("val="+val);  
    }  
    catch(NumberFormatException e) {  
        e.printStackTrace();  
    }  
}
```



Lorsqu'il y a erreur de conversion dans **convertir()**, l'objet Exception **e** est remonté vers la méthode appelante **traiterNombre()** ; cette dernière présente la capture (blocs try et catch) de l'erreur. C'est donc cette méthode qui intercepte l'exception.

Dans le cas où aucune méthode n'a prévu de gérer l'exception, le programme est arrêté.

Interceptor plusieurs exceptions



```
public void traiterNombre() {
    Scanner sc=new Scanner(System.in);
    int [] tab = null;
    try{
        String ch = sc.next(); // Saisie d'une chaine
        int a = convertir(ch);
        ch = sc.next(); // Saisie d'une nouvelle chaine
        int b = convertir(ch);
        int val = a / b ;
        tab[0]=val;
        System.out.println("Tout s'est bien passé !!");
    }
    catch(NumberFormatException e) {
        e.printStackTrace();
    }
    catch(ArithmeticException e) {
        e.printStackTrace();
    }
    catch(NullPointerException e) {
        e.printStackTrace();
    }
    catch(Exception e) {
    }
}
```

- Il est tout a fait possible d'utiliser plusieurs gestionnaires d'exceptions.
- Cela permet de traiter différents types de problèmes de manière séparée.
- Il suffit de mettre autant de blocs catch que nécessaire.
- Java évalue du haut vers le bas le gestionnaire susceptible de traiter l'exception.

Traitement de l'erreur de conversion

Traitement de l'erreur de calcul

Traitement de l'erreur d'utilisation d'une référence nulle

Traitement de l'erreur générique

Attention : ne pas mettre l'exception générique avant les spécifiques

Déclencher une exception : throw



- Pour déclencher une exception on utilise le mot **throw** (sans s à la fin)
- La méthode qui déclenche l'exception doit le préciser dans sa signature par le mot **throws** (avec s à la fin).
- Quand dans une méthode on déclenche une exception avec throw, la méthode sera interrompue et l'objet exception sera remonté à la méthode appelante.

```
public class ExceptionExample {  
    public static char charAt(char[] chaine,int index) {  
        if (index<0 || index>=chaine.length)  
            throw new IllegalArgumentException("index incorrect : "+index);  
        return chaine[index];  
    }  
    public static void main(String[] args) {  
        String ch="Bonjour les amis";  
        char[] array=ch.toCharArray();    //convertit la chaine en un  
                                           //tableau de caractères  
  
        charAt(array,0);  
        charAt(array,30);  
    }  
}
```

La fonction charAt sera interrompue si index dépasse la taille de chaine ou si la chaine est vide.

throw permet d'instancier un objet de type IllegalArgumentException

Console

```
<terminated> ExceptionExample [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  
Exception in thread "main" java.lang.IllegalArgumentException: index incorrect : 30  
    at ExceptionExample.charAt(ExceptionExample.java:6)  
    at ExceptionExample.main(ExceptionExample.java:13)
```

Propager une exception : throws



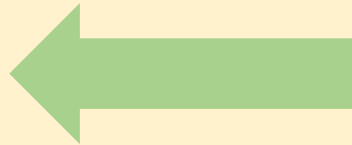
```
public static char charAt(char[] array,int index) {
    if (index<0 || index>=array.length)
        throw new IllegalArgumentException("index incorrect : "+index);
    return array[index];
}
public static void saisie() throws IllegalArgumentException{
    Scanner sc = new Scanner(System.in);
    System.out.println("Saisir une chaine de caractères : ");
    String ch = sc.next();
    char[] array=ch.toCharArray();
    char c = charAt(array,0);
    c = charAt(array,30);
}
public static void main(String[] args) {
    try{
        saisie();
    }
    catch(IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }
}
```

La fonction `saisie()` ne traite pas l'exception générée par `charAt`. Elle la propage à la fonction appelante (ici `main()`) en utilisant **throws**.

S'il y a erreur, la fonction est interrompue, et l'exception est propagée.

C'est dans le `main` que l'exception est traitée dans un bloc `try` et `catch`.

La méthode `getMessage()` récupère l'information dans le bloc `catch`



Définir sa propre exception



- Pour créer sa propre classe d'exception, il suffit donc de créer une classe qui hérite de **java.lang.Exception**
- La plupart du temps, on crée une nouvelle exception pour
 - gérer, par programmation (et non par message d'erreur), les utilisations incorrectes d'une méthode.
 - prévenir l'utilisateur de la méthode dont certaines utilisations sont incorrectes et qu'elles doivent être gérées explicitement.

```
//Définition de l'exception
class AgeInvalidException extends Exception{
    AgeInvalidException(String s){
        super(s);
    }
}

public class ExceptionExample {
    static void check(int age) throws AgeInvalidException{
        if(age < 20)
            throw new AgeInvalidException("Pas valide");
        else
            System.out.println("Valide");
    }
    public static void main(String args[]){
        try{
            check(15);
        }
        catch(Exception ex){
            System.out.println("Une exception s'est produite: " + ex.getMessage());
        }
        System.out.println("...");
    }
}
```

L'exception spécifique de type `AgeInvalidException` sera interceptée dans le main ou encore sous forme générique (`Exception`).

```
terminated> ExceptionExample [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (25 oct. 2021 à 15:00)
Une exception s'est produite: AgeInvalidException: Pas valide
...
```

Autre exemple de définition de classe d'exception



```
//Définition de l'exception
class LargeListException extends RuntimeException{

    public LargeListException(String message) {
        super(message);
    }
}

//Classe qui utilise l'exception définie
public class Stock {

    public void check(List<String> products) {
        if (products.size() > 10) {
            throw new LargeListException("La liste ne peut dépasser 10 produits!");
        }
    }

    public static void main(String[] args) {
        Stock stock = new Stock();
        List<String> products = new ArrayList<>();
        for(int i = 0; i < 20; i++){
            products.add("Product"+i);
        }

        stock.check(products);
    }
}
```

Console

<terminated> Stock [Java Application] C:\dev22\eclipse-p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\

Exception in thread "main" [LargeListException](#): La liste ne peut dépasser 10 produits!

at Stock.check([Stock.java:17](#))

at Stock.main([Stock.java:33](#))

Bloc finally



- Le block **finally** est un block optionnel contenant des instructions qui seront toujours exécutées quelque soit le résultat du try.
- Cela permet de s'assurer que les ressources éventuellement utilisées sont toujours libérées.

```
public static void main(String args[]){  
    Scanner sc=new Scanner(System.in);  
    try{  
        System.out.println("Donner un entier : ");  
        int i=sc.nextInt();  
    }  
    catch(Exception e){  
        System.out.println("Saisie invalide :"+  
            e.getMessage());  
    }  
    finally{  
        System.out.println("Fermeture de l'objet Scanner.");  
        sc.close();  
    }  
}
```

Quelque soit la saisie (saisie correcte ou incorrecte), le bloc **finally** sera exécuté.

Le try-with-resources



- On peut utiliser la syntaxe try-with-resources avec les objets qui implémentent l'interface **java.lang.AutoCloseable**.

```
static String readFirstLineFromFile(String path) throws IOException{  
    try(BufferedReader br=new BufferedReader(new FileReader(path))){  
        return br.readLine();  
    }  
}
```

- `Java.io.BufferedReader` implémente `java.lang.AutoCloseable`
- On voit ici que le bloc catch est optionnel

Un seul bloc catch / plusieurs exceptions



- Permet de limiter la redondance dans le code
- Réduit la tentation d'utiliser **java.lang.Exception**

```
public static void main(String args[]){  
    try {  
        BufferedReader br=new BufferedReader(new FileReader("chemin"));  
        Connection conn = DriverManager.getConnection("chaîne de connexion");  
    }  
    catch(IOException | SQLException ex) {  
        ex.printStackTrace();  
    }  
}
```