

Algorithmique et programmation par objets

Inf F3

Licence 2 MIASHS

Université Grenoble Alpes

Jerome.David@univ-grenoble-alpes.fr

2024-2025

<http://miashs-www.u-ga.fr/~davidjer/inff3/>

Cours 8 : Les exceptions

- « Mieux prévenir que guérir »
 - Java fourni de nombreux moyen de prévenir les erreurs (typage statique, constructeurs, etc.)
 - Mais, on ne peut pas tout contrôler
 - Cela prend un temps considérable,
 - Et on ne peut pas penser à tout
- « Qui ne risque rien n'a rien »
 - On prend parfois des risques (inconsciemment ou non) :
 - transtypage (descendant ou restrictif), conversion chaînes vers nombre, accès au réseau, lecture de fichiers, etc.
- Afin de réaliser un bon compromis entre prise de risque et robustesse du code, Java offre un mécanisme de récupération sur erreur performant :

Les Exceptions

Code risqué

- Dans de nombreuses situations, on est amené à écrire du code risqué

```
public class Dictionnaire {  
    private String[] mots;  
    private String[] definitions;  
  
    public Dictionnaire(int capaciteInitiale) {  
        mots = new String[capaciteInitiale];  
        definitions = new String[capaciteInitiale];  
    }  
    // ...  
}
```

```
public class IntroExceptions {  
    public static void main(String[] args) {  
        Dictionnaire dico = new Dictionnaire(-2);  
    }  
}
```

Que donne l'exécution de la classe IntroExceptions ?

```
Exception in thread "main" java.lang.NegativeArraySizeException  
    at cours9.Dictionnaire.<init>(IntroExceptions.java:9)  
    at cours9.IntroExceptions.main(IntroExceptions.java:16)
```

Code risqué

```
public class Dictionnaire {  
    private String[] mots;  
    private String[] definitions;  
  
    public Dictionnaire(int capaciteInitiale) {  
        mots = new String[capaciteInitiale];  
        definitions = new String[capaciteInitiale];  
    }  
    public String getMot(int idx) {  
        return mots[idx];  
    }  
    // ...  
}
```

```
public class IntroExceptions {  
    public static void main(String[] args) {  
        Dictionnaire dico = new Dictionnaire(2);  
        System.out.println(dico.getMot(2));  
    }  
}
```

Et dans ce cas la ?

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2  
    at cours9.Dictionnaire.getMot(IntroExceptions.java:14)  
    at cours9.IntroExceptions.main(IntroExceptions.java:22)
```

Les exceptions

- Comment savoir qu'une méthode que j'appelle est risquée ?

- Exemple : `int i = Integer.parseInt("Je ne suis pas un entier");`

- <https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#parseInt%28java.lang.String%29>

parseInt

```
public static int parseInt(String s)  
    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value or an ASCII plus sign '+' ('\u002B') to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

Parameters:

s - a String containing the int representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

`NumberFormatException` - if the string does not contain a parsable integer.

Alors qu'est ce qu'une exception ?

- Une exception est une sorte de « condition exceptionnelle »
 - Un problème qui empêche la continuation de la méthode ou du bloc courant
 - Parce que l'on ne dispose pas d'information suffisante dans le contexte courant pour traiter ce problème
- La solution des exceptions est reléguer le problème au niveau supérieur (i.e. au bloc appelant)

Situations exceptionnelles ?

- Exemple : la division par 0 en Java lève une exception :
 - La moyenne d'un étudiant est la moyenne de ses notes, si il n'a pas de note alors sa moyenne est 0
 - Dans ce cas, il faut traiter le cas de 0 notes

```
class Etudiant {  
    private int sommeNotes;  
    private int nbNotes;  
  
    public void addNote(int note) {  
        sommeNotes+=note;  
        nbNotes++;  
    }  
    public double moyenne() {  
        if (nbNotes==0) {  
            return 0;  
        }  
        return ((double) sommeNotes)/nbNotes;  
    }  
}
```

Situations exceptionnelles ?

- La moyenne d'une promo est la moyenne de ses étudiants
 - Une promo où il y a 0 étudiants n'est pas un cas envisagé.
 - C'est une situation exceptionnelle : on laisse Java lever l'exception si ce cas se présente

```
class Promo {  
    private Etudiant [] etudiants;  
    // ...  
  
    public double moyennePromo() {  
        double somme=0;  
        for (Etudiant e : etudiants) {  
            somme+=e.moyenne();  
        }  
        return somme/etudiants.length;  
    }  
}
```


Exceptions et Java

- En java les Exceptions sont des objets
- Java fourni des mots clés pour
 - Lever une exception (via le mot-clé `throw`)
 - On veut signaler une condition exceptionnelle à la méthode appelante
 - Si on lève une exception l'exécution du code est interrompue
 - Intercepter une exception (via le mot-clé `catch`)
 - Permet d'exécuter les actions « correctives » permettant de gérer cette situation exceptionnelle
- Qu'est ce qui se passe si l'on n'intercepte pas l'exception ?
 - Le programme s'arrête avec un beau message d'erreur comme vous avez déjà vu

Levée d'exception

```
class Parachute {  
    public void ouvrir() {  
        // ...  
    }  
}
```

```
class Parapentiste {  
    private Parachute parachute;  
  
    public void acheterUnParachute(Parachute p) {  
        parachute=p;  
    }  
  
    public void sauter() {  
        if (parachute==null) {  
            throw new RuntimeException("Aie !!!");  
        }  
        parachute.ouvrir();  
    }  
}
```

Ligne 60

```
class CoupeIcare {  
    public void parade(Parapentiste[] participants) {  
        for (Parapentiste p : participants) {  
            p.sauter();  
        }  
    }  
}
```

Ligne 70

```
public class IntroExceptions {  
    public static void main(String[] args) {  
        Parapentiste casseCou = new Parapentiste();  
        Parapentiste inconscient = new Parapentiste();  
        Parapentiste prevoyant = new Parapentiste();  
  
        Parapentiste[] participants = new Parapentiste[] {prevoyant,casseCou,inconscient};  
  
        CoupeIcare c = new CoupeIcare();  
  
        c.parade(participants);  
    }  
}
```

Ligne 85

Une exception est toujours relancée à l'appelant : c'est le principe de la « patate chaude »

```
Exception in thread "main" java.lang.RuntimeException:  
Aie !!!  
    at cours9.Parapentiste.sauter(IntroExceptions.java:60)  
    at cours9.CoupeIcare.parade(IntroExceptions.java:70)  
    at cours9.IntroExceptions.main(IntroExceptions.java:85)
```

Interception d'exceptions

Les exceptions peuvent (doivent)
être interceptées dans un des
blocs appelant
→ via les blocs try/catch

```
class CoupeIcare {  
    public void parade(Parapentiste[] participants) {  
        for (Parapentiste p : participants) {  
            p.sauter();  
        }  
    }  
}
```

```
class Parapentiste {  
    private Parachute parachute;  
  
    public void acheterUnParachute(Parachute p) {  
        parachute=p;  
    }  
  
    public void sauter() {  
        if (parachute==null) {  
            throw new RuntimeException("Aie !!!");  
        }  
        parachute.ouvrir();  
    }  
}
```

Exercice : Dans le cas présenté, la parade est arrêté en cas d'accident. Comment faire pour qu'elle puisse continuer (tout en traitant l'exception et en appelant les secours) ?

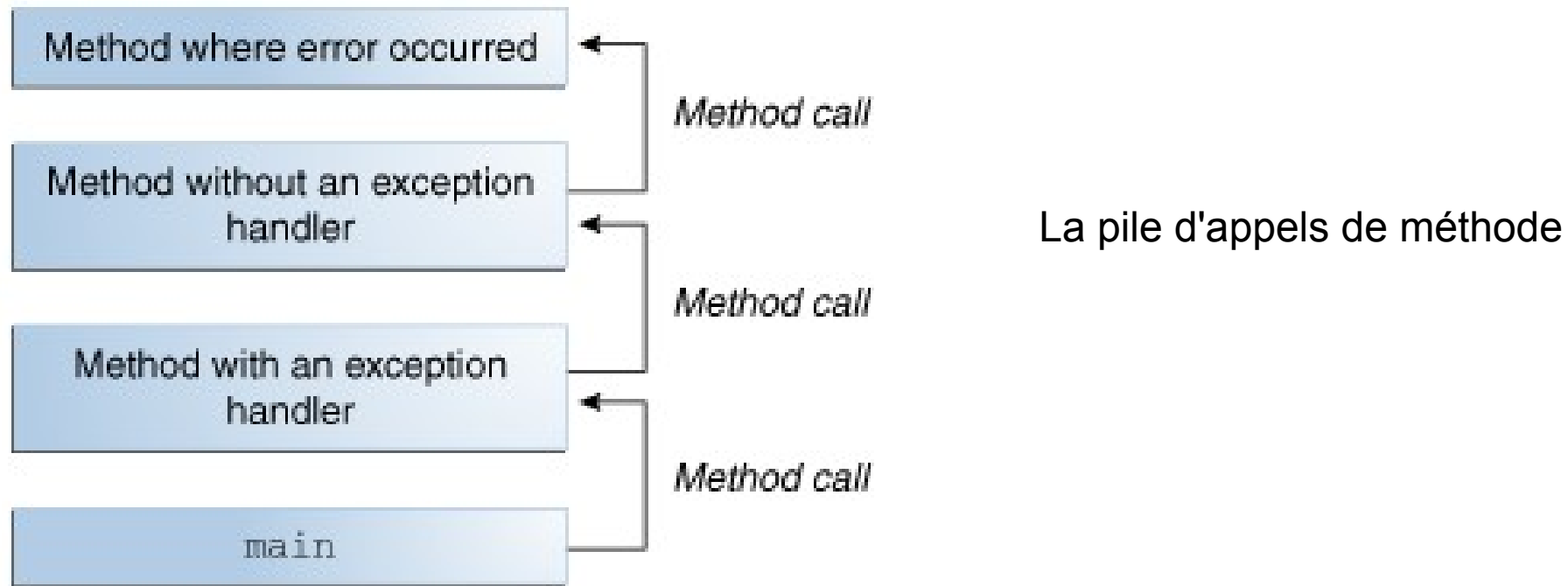
```
public class IntroExceptions {  
    public static void main(String[] args) {  
        Parapentiste casseCou = new Parapentiste();  
        Parapentiste inconscient = new Parapentiste();  
        Parapentiste prevoyant = new Parapentiste();  
  
        Parapentiste[] participants = new Parapentiste[] {prevoyant, casseCou, inconscient};  
  
        CoupeIcare c = new CoupeIcare();  
        try {  
            c.parade(participants);  
            // voter pour élire le gagnant  
            System.out.println("Le gagnant est ...");  
        }  
        catch (RuntimeException e) {  
            System.err.println("Il faut appeler les secours !!!");  
        }  
    }  
}
```

Il faut appeler les secours !!!

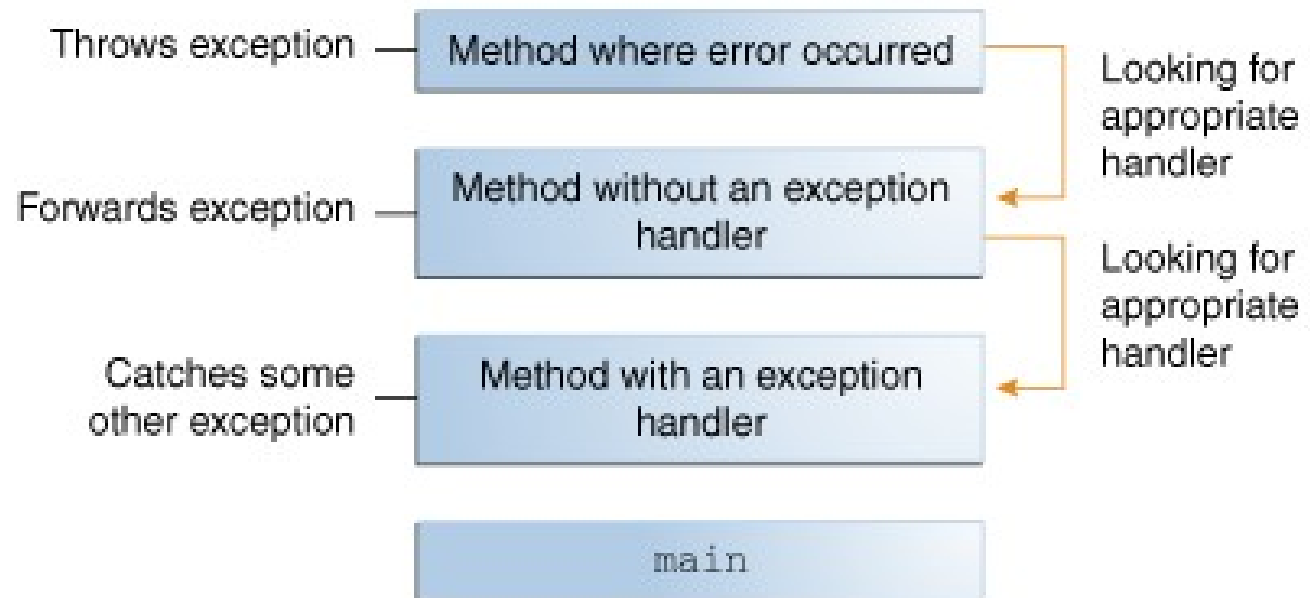
Résumé du fonctionnement

- Une exception est levée via le mot-clé `throw`
- Elle se propage dans la pile des méthodes appelantes tant qu'elle n'est pas interceptée
 - Bloc try/catch correspondant à l'exception
 - Si elle se propage après la méthode main, alors la trace de la pile des appels est affichée (message d'erreur) et l'exécution du programme s'arrête

Résumé - graphique



Ordre de recherche d'un intercepteur d'exception (bloc try/catch) approprié



L'objet «Throwable»

- Lors d'une exception, la communication avec les blocs appelant se fait via un objet représentant l'exception
 - En java les Exceptions sont des objets
 - `throw new RuntimeException()`
 - Il existe une hiérarchie (d'héritage) des exceptions
 - La racine de l'héritage des exceptions est la classe `java.lang.Throwable`

Héritage des Exceptions

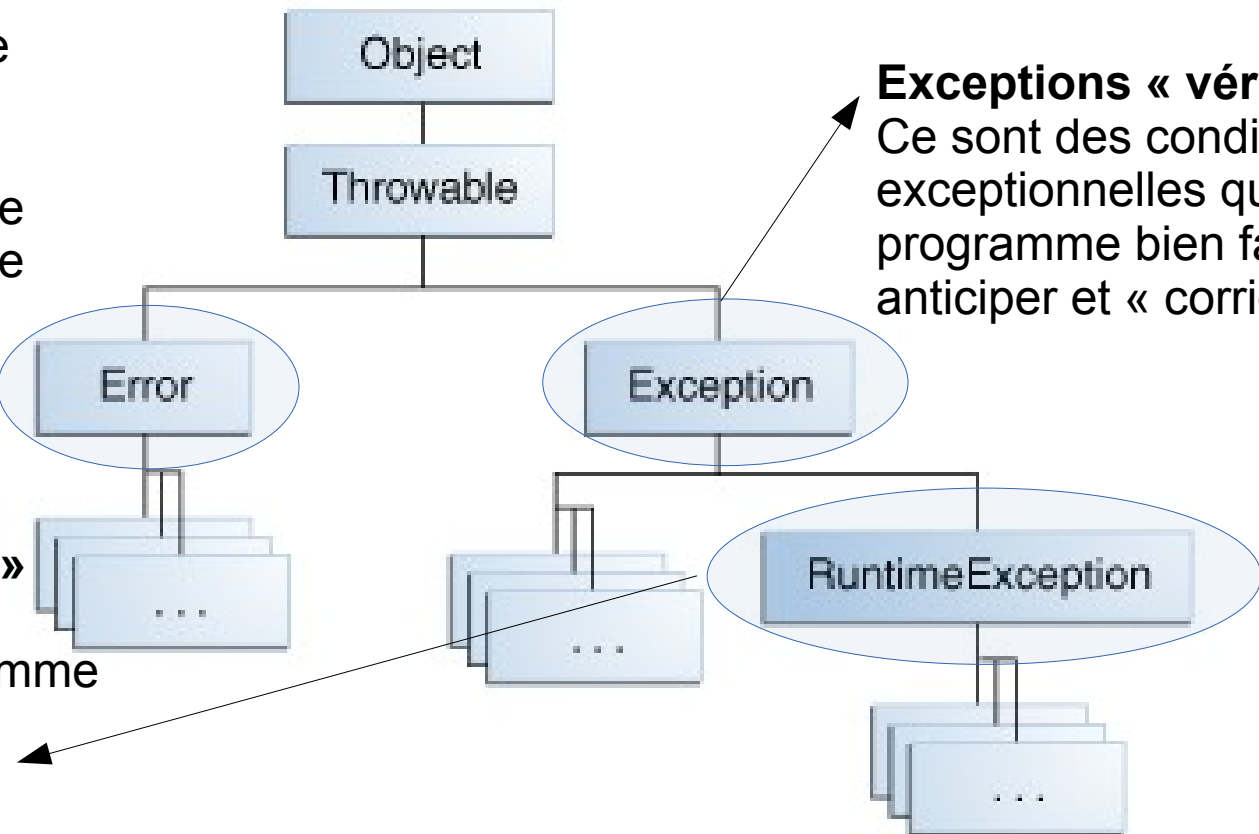
- Il existe trois types principaux d'exceptions :

Les erreurs

Ce sont des situations externe à l'application que l'on ne peut anticiper et « corriger ».
→ exemple : une erreur de lecture sur un périphérique

Exceptions « vérifiées »

Ce sont des conditions exceptionnelles qu'un programme bien fait doit anticiper et « corriger ».



Exceptions non « vérifiées »
Ce sont des conditions exceptionnelles qu'un programme ne peut pas anticiper et « corriger ». Elle indiquent généralement une mauvaise utilisation de la méthode appelée (bug)

L'obligation d'interception et propagation explicite

- Les exceptions « vérifiées » sont soumises à l'obligation d'interception ou de propagation explicite.
- Retour au parachutiste...

```
public void sauter() {  
    if (parachute==null) {  
        throw new Exception("Aie !!!");  
    }  
    parachute.ouvrir();  
}
```

Unhandled exception type Exception

2 quick fixes available:

- [Add throws declaration](#)
- [Surround with try/catch](#)

Spécification de propagation à l'appelant

```
public void sauter() throws Exception {  
    if (parachute==null) {  
        throw new Exception("Aie !!!");  
    }  
    parachute.ouvrir();  
}
```

```
public void parade(Parapentiste[] participants) {  
    for (Parapentiste p : participants) {  
        try {  
            p.sauter();  
        } catch (Exception e) {  
            System.err.println(e.getMessage());  
            // appeler les secours !  
        }  
    }  
}
```

On lève une exception vérifiée, on est donc soumis à l'obligation d'interception ou de propagation explicite

Interception d'exceptions

- Les exceptions doivent être en général « interceptées »
- Pour cela, il faut utiliser la structure suivante :

```
try {  
    // code susceptible de lever une exception  
}  
catch (Exception1 e1) {  
    // code permettant de traiter une exception  
    // du type Exception1  
}  
catch (Exception2 e2) {  
    // code permettant de traiter une exception  
    // du type Exception2  
}  
...
```

Exemple d'interception

- Un programme qui compte les lignes d'un fichier

```
import java.io.*;

public class LineCount {
    public static void main(String[] args) {
        File f = new File(args[0]);
        int nbLines=0;
        try {
            RandomAccessFile r = new RandomAccessFile(f,"r");
            for (String line=r.readLine() ; line!=null; line=r.readLine()) {
                nbLines++;
            }
            r.close();
        } catch (FileNotFoundException e1) {
            System.err.println("Le fichier "+f+" n'existe pas");
        } catch (IOException e2) {
            System.err.println("Il y a eu une erreur à la lecture du fichier : "
                               +e2.getMessage());
        }
        System.out.println(nbLines);
    }
}
```

java -cp bin LineCount src/Point.java
22

Si je ne passe aucun argument, j'obtiens ce message :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at cours9.LineCount.main(LineCount.java:5)
```

Comment traiter cette exception pour avoir un message compréhensible ?

Sujets non abordés

- Comment la clause catch est sélectionnée ?
 - En cas d'héritage entre exception
- Le bloc finally
- Les nouveautés java7 sur les exceptions
- Le chaînage d'exceptions
- Le choix entre RuntimeException ou Exceptions

Exercices

- Exercices
 - Modifier cette classe pour qu'elle lève une Exception (type `RuntimeException`) quand l'âge est < 0

```
public class Personne {  
    private String prenom;  
    private String nom;  
    private int age;  
  
    public Personne(String p , String n, int a) {  
        nom=n;  
        prenom=p;  
        age=a;  
    }  
    public String toString() {  
        return "Je m'appelle "+nom+" "+prenom+" et j'ai "+age+" ans";  
    }  
}
```