Gestion des exceptions en Java

Jean-Marie Lagniez

LPDIOC : Algorithmique et Program mation
IUT Lens

Département Informatique

Table des matières

Sommaire

Dans ce cours nous allons voir :

- Qu'est ce qu'une exception ?
- Comment capturer et manipuler les exceptions
- Comment lancer une exception
- Les exceptions et la gestion des ressources (I/O)
- Le non traitement des exceptions
- Exemple d'utilisation

Pré-requis :

- Héritage, création d'objets, polymorphisme et classes abstraites

C'est quoi une exception ?

- Le terme *exception* est un raccourci pour *exception event*
- **Formellement, une exception est un événement qui apparaît durant l'exécution du programme et qui perturbe le déroulement de celui-ci**
- Lorsqu'une erreur survient dans une méthode alors cette méthode **génère un objet** et l'envoie sur **la pile d'exécution du programme** :

  $\Rightarrow$ on dit que la méthode a soulevé une exception

- Cet objet, appelé en anglais *exception object*, inclue dans sa définition son type et l'état du programme lorsque l'erreur est survenu
- Créer une exception et l'envoyer sur la pile d'exécution est appelé **jeter une exception** (ou en anglais *throwing an exception*)
- Cette exception pourra être gérer par une sous-routine : ***handler***

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

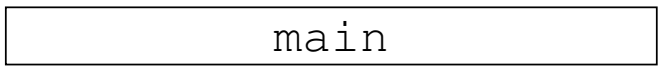


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

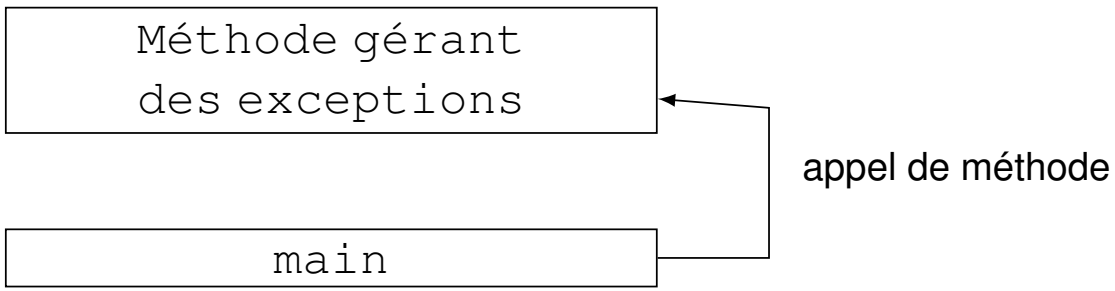


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

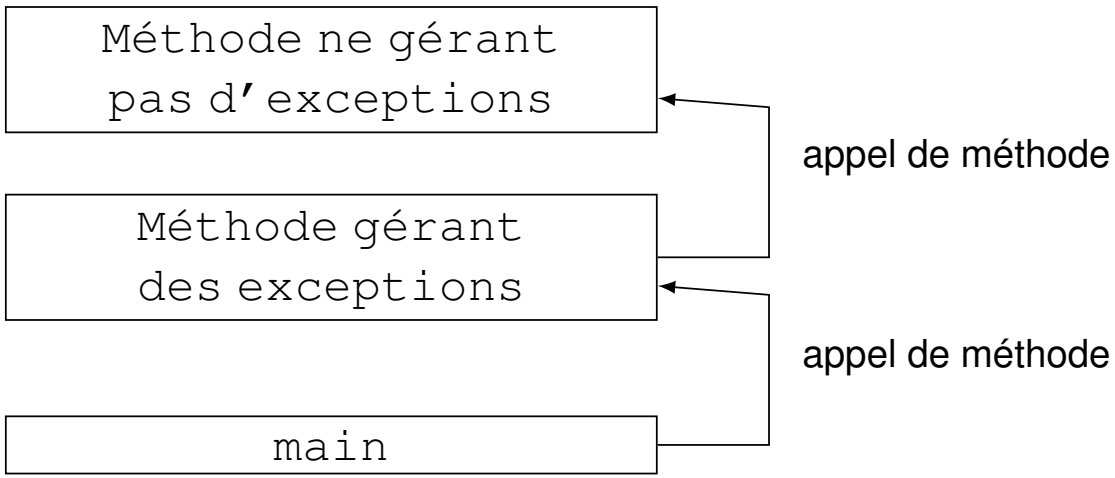


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

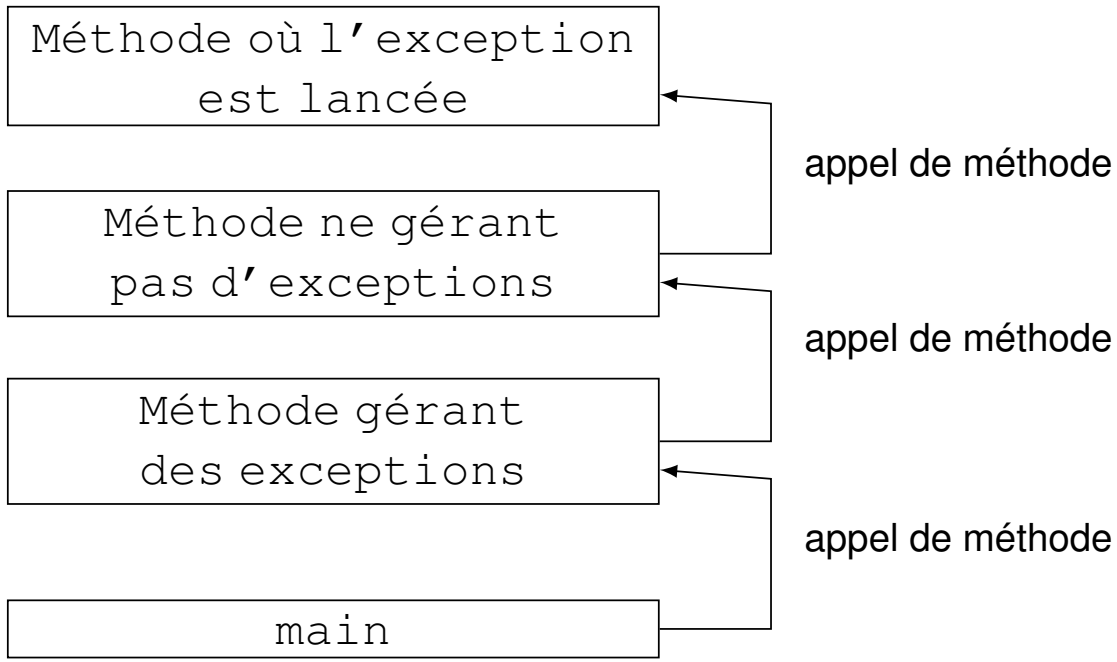


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

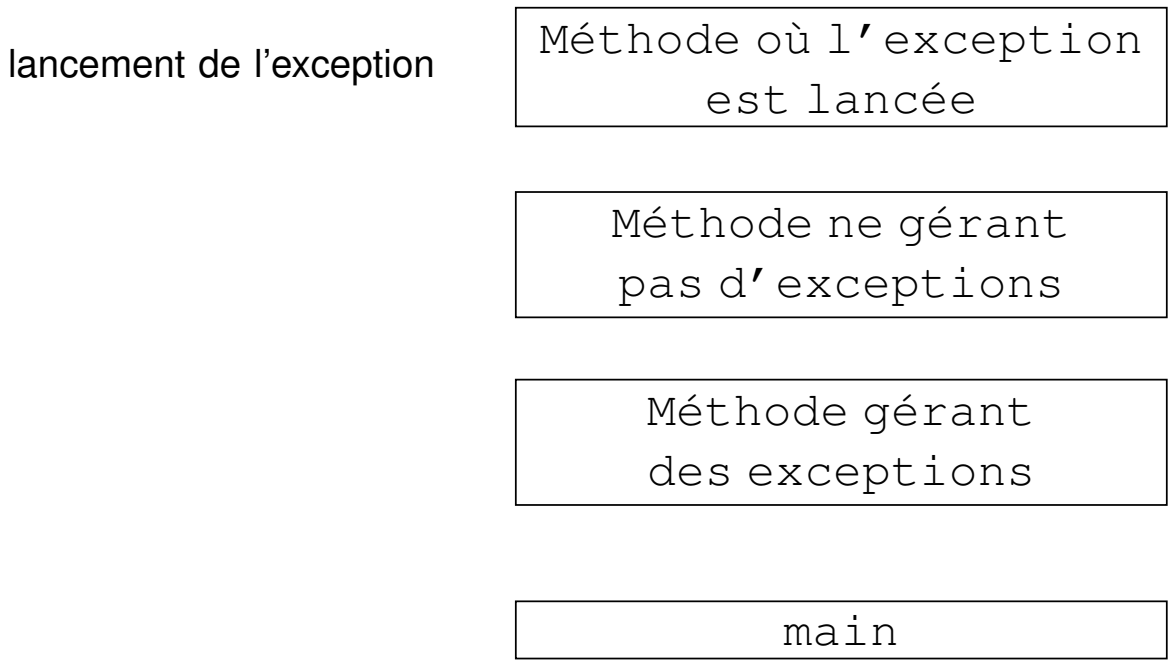


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

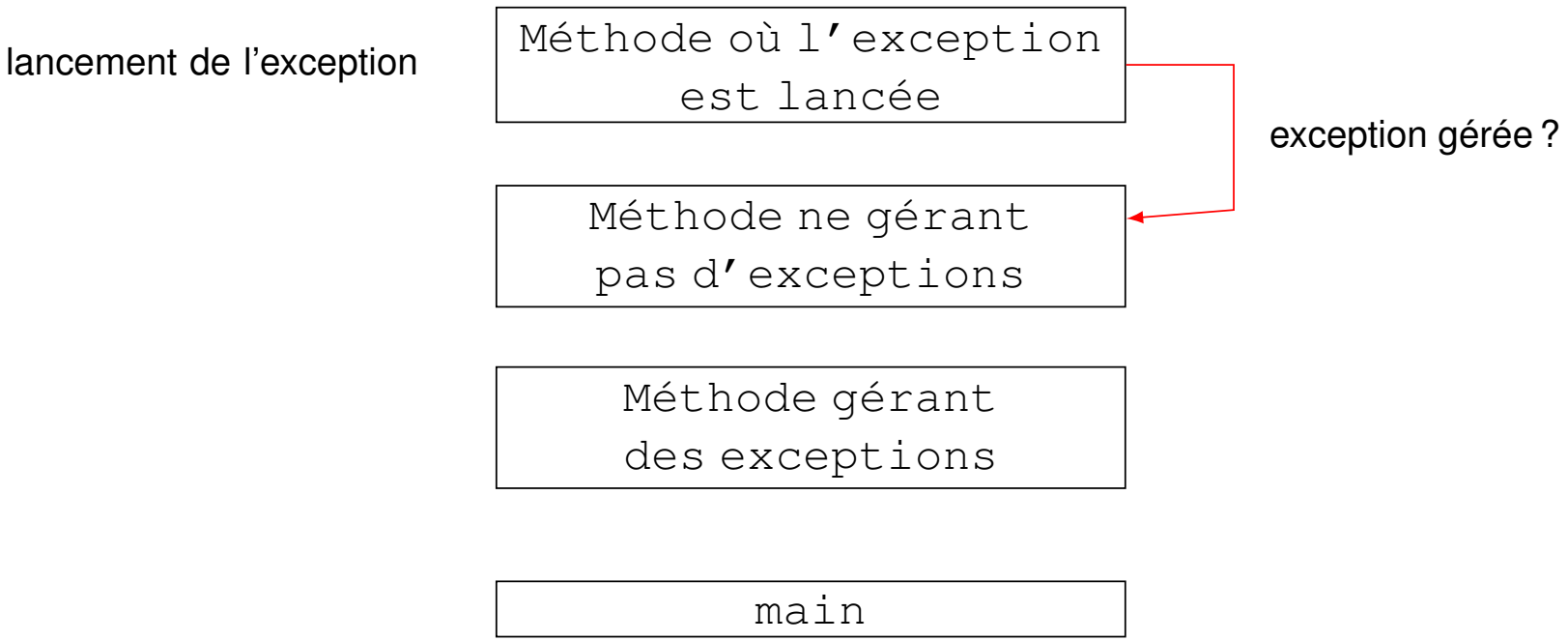


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

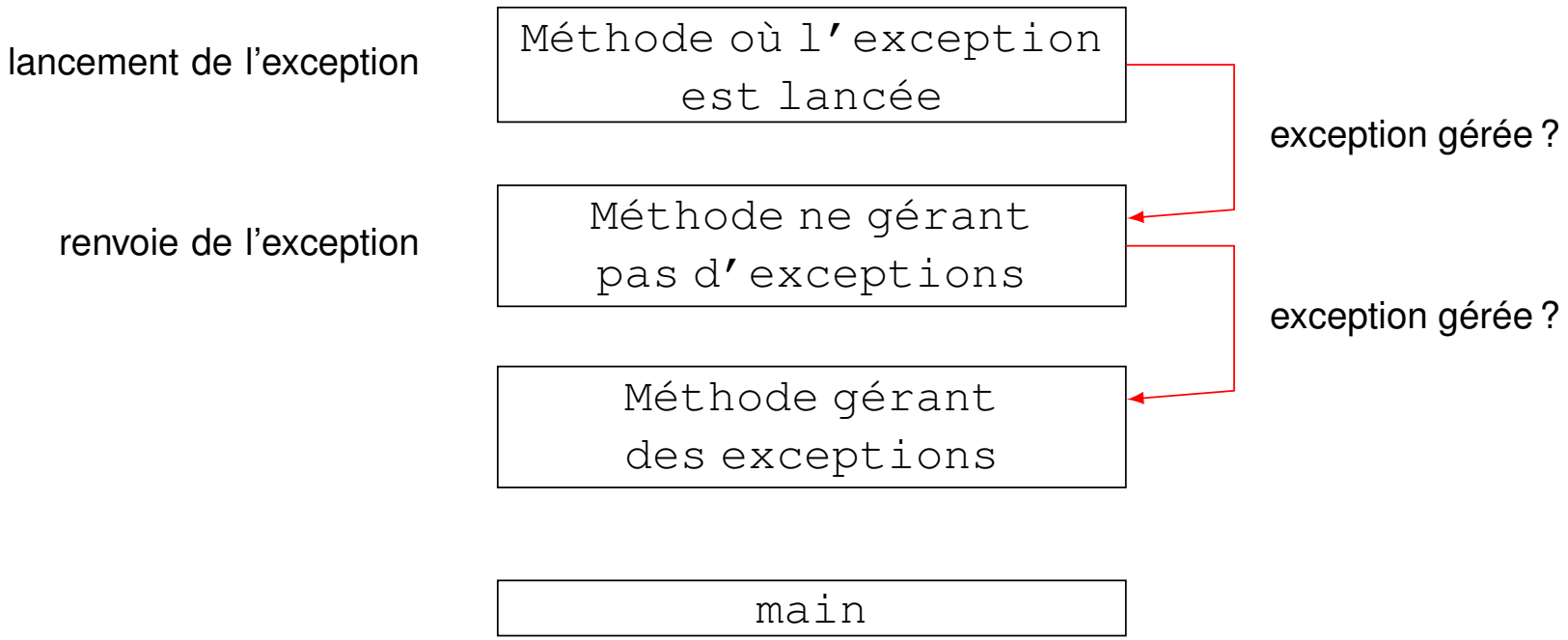


FIGURE – La pile d'exécution

Propagation et gestion des exceptions

- Lorsqu'une exception est jeter le système essaie de trouver un moyen de la gérer
- Pour cela, il va remonter la **pile d'exécution du programme** afin de trouver une méthode qui puisse **capturer** cette dernière

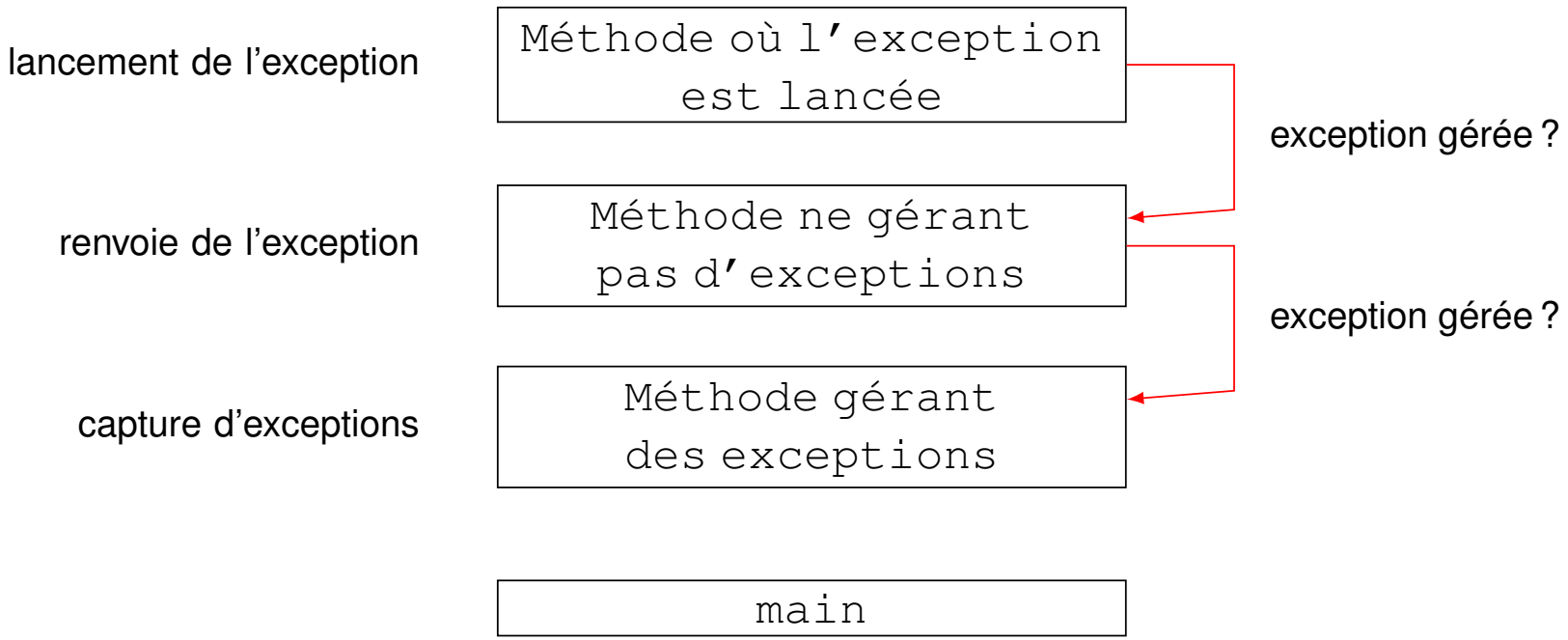


FIGURE – La pile d'exécution

Gotta Catch 'Em All

- Un **programme** est **valide** si et seulement si il s'assure que lorsqu'une **méthode spécifie formellement qu'elle est susceptible de lancer une exception** alors une procédure est mise en place pour **traiter cette exception**
- Cela signifie qu'une méthode pouvant lancer une exception se trouvera placer dans un bloc particulier : **un bloc try {} catch {}** en Java
- Il faut aussi pouvoir faire en sorte de spécifier qu'une méthode peut lancer une exception via l'utilisation par exemple d'un mot clef : **throws** en Java

- Toutes les exceptions ne sont pas vouées à être intercepté. **On considère trois catégories d'exceptions** :
  - Les exceptions que l'on peut anticiper car bien identifiées dans l'application (mauvaise saisie d'un nom de fichier)
  - Les exceptions provenant d'erreurs externes à l'application (lecture non autorisée d'un fichier)
  - Les exceptions interne à l'application mais qui ne peuvent pas être anticipées (accès à un élément à l'extérieure de la zone du tableau)

Table des matières

Faut-il avoir confiance en l'utilisateur ?

```
import java.util.Scanner;
import java.util.*;
class Test{
  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.print("Numérateur : ");
    int v1 = sc.nextInt();
    System.out.print("Dénominateur : ");
    int v2 = sc.nextInt();
    System.out.println(v1 + "/" + v2 + "=" + v1/v2);
  }
}
```

- Si l'utilisateur n'entre pas une valeur correcte alors une exception peut être levée. Ainsi le dénominateur vaut 0 alors le programme termine en jetant une exception `java.lang.ArithmeticException`

```
Exception in thread "main"
    java.lang.ArithmeticException: / by zero
      at Test.main(Test.java:14)
```

Le bloc *try*

- La première étape lorsque l'on souhaite capturer une exception est de mettre le code sensible dans un bloc **try**

```
try{
  // code
}
catch et finally ...
```

- Le segment représenté par le commentaire dans le code ci-dessus contient une ou plusieurs lignes de codes valides pouvant générer une exception

```
public static void main(String[] args){
  Scanner sc = new Scanner(System.in);
  System.out.print("Numérateur : ");
  int v1 = sc.nextInt();
  System.out.print("Dénominateur : ");
  int v2 = sc.nextInt();
  try{
    System.out.println(v1 + "/" + v2 + "=" + v1/v2);
  }
  catch et finally ...
}
```

Le bloc *catch*

- La procédure à suivre lorsqu'une exception est levée est donnée via l'utilisation du mot clef **catch**

```
try{
  // code
} catch (ArithmeticException e) {
  System.err.println("Capture de ArithmeticException: " +
      e.getMessage());
} catch (InputMismatchException e) {
  System.err.println("Capture de IOException: " +
      e.getMessage());
}
```

- Depuis la version 7 de Java il est possible de capturer plusieurs exceptions avec un seul bloc

```
try{
  // code
} catch (ArithmeticException|InputMismatchException e) {
  System.err.println("Exception: " + e.getMessage());
}
```

- Si plusieurs exception sont capturées avec un seul `catch` alors le paramètre `e` est implicitement `final`

Le bloc *finally*

- En Java il est possible de forcer l'exécution d'un bout de code grâce au mot clef **finally**

```
try{
  // code
} catch ... {
  // code pour le traitement de l'exception
} finally {
  // code systématiquement exécuté
}
```

- Ainsi quelque soit le code qui sera exécuter la partie du code se trouvant dans le bloc `finally` sera exécuté

- L'utilisation du bloc `finally` peut par exemple être utilisé lorsqu'une application nécessite de manipuler des fichiers. En effet, dans ce cas les fichiers doivent nécessairement être fermés à la fin de l'application

Exercice

- Modifiez le code suivant afin de vous assurer que l'utilisateur ne fasse pas planter la machine. Dans tous les cas il faut lui dire au revoir :D

```
public static void main(String[] args){
  Scanner sc = new Scanner(System.in);
  System.out.print("Numérateur : ");
  int v1 = sc.nextInt();
  System.out.print("Dénominateur : ");
  int v2 = sc.nextInt();
  System.out.println(v1 + "/" + v2 + "=" + v1/v2);
}
```

Exercice

- Modifiez le code suivant afin de vous assurer que l'utilisateur ne fasse pas planter la machine. Dans tous les cas il faut lui dire au revoir :D

```
public static void main(String[] args){
  Scanner sc = new Scanner(System.in);
  try{
    System.out.print("Numérateur : ");
    int v1 = sc.nextInt();
    System.out.print("Dénominateur : ");
    int v2 = sc.nextInt();
    System.out.println(v1 + "/" + v2 + "=" + v1/v2);
  }catch(Exception e){
    System.err.println("Un problème est survenu : "
        + e.getMessage());
  }finally{
    System.out.println("Au revoir");
  }
}
```

Table des matières

Spécifier les exceptions lancées par une méthode

- Nous avons vu précédemment comment capturer et gérer des exceptions
- En pratique on utilise des classes codées par d'autres personnes et donc des méthodes pouvant être dans un état incorrect par les paramètres passés

```
import java.util.Scanner;
import java.util.*;
class Test{
  public int div(int a, int b){
    return a / b; // si b = 0 on a un problème !
  }

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.print("Numérateur : ");
    int v1 = sc.nextInt();
    System.out.print("Dénominateur : ");
    int v2 = sc.nextInt();
    System.out.println(v1 + "/" + v2 + "=" + div(v1, v2));
  }
}
```

- Il est donc nécessaire de prévenir l'utilisateur qu'une exception peut être levée par une méthode : utilisation du mot clef **throws**

Spécifier les exceptions lancées par une méthode

- Nous avons vu précédemment comment capturer et gérer des exceptions
- En pratique on utilise des classes codées par d'autres personnes et donc des méthodes pouvant être dans un état incorrect par les paramètres passés

```
import java.util.Scanner;
import java.util.*;
class Test{
  public int div(int a, int b) throws ArithmeticException{
    // vérifier b et lancer l'exception si nécessaire !
  }

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.print("Numérateur : ");
    int v1 = sc.nextInt();
    System.out.print("Dénominateur : ");
    int v2 = sc.nextInt();
    System.out.println(v1 + "/" + v2 + "=" + div(v1, v2));
  }
}
```

- Il est donc nécessaire de prévenir l'utilisateur qu'une exception peut être levée par une méthode : utilisation du mot clef **throws**

Lancer une exception

- Avant de pouvoir être capturée une exception dans être lancée
- Une méthode qui souhaite lancer une exception utilisera le mot clef **throw**

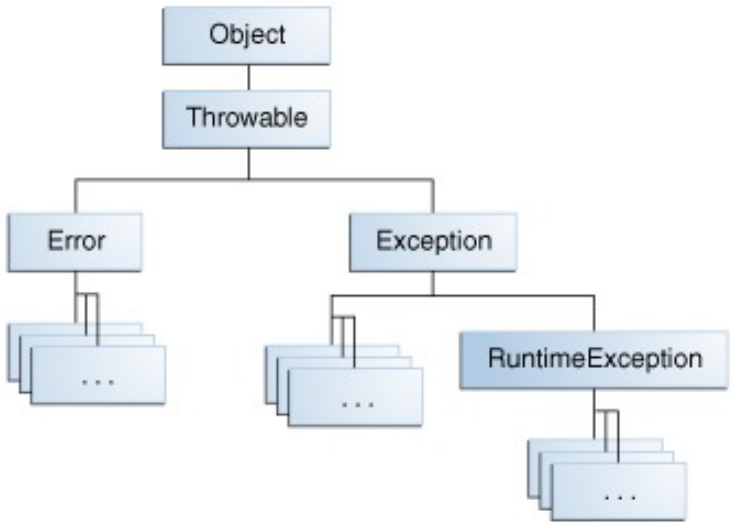
```
import java.util.Scanner;
import java.util.*;
class Test{
  public int div(int a, int b) throws ArithmeticException{
    if(b == 0) throw new ArithmeticException();
    return a/b;
  }

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.print("Numérateur : ");
    int v1 = sc.nextInt();
    System.out.print("Dénominateur : ");
    int v2 = sc.nextInt();
    System.out.println(v1 + "/" + v2 + "=" + div(v1, v2));
  }
}
```

- **Il est possible de créer ses propres exception !**

Table des matières

La classe `Throwable` et ses sous-classes

- Les objets qui hérite de la classe `Throwable` se divise en deux branches



- Classe `Error` : ce sont des problèmes rencontrés le plus souvent au niveau de la JVM (un programme simple ne lancera pas d'objet de type `Error`)
- Classe `Exception` : la plupart des objets capturés pas une application sont de ce type. Une exception indique qu'un problème est survenu mais que ce n'est pas un problème "trop" sérieux

Création d'un objet de type `Exception`

- Pour créer sa propre exception nous allons simplement créer un objet qui hérite de la classe `Exception`

```
class EtudiantPasContentException extends Exception{
  public EtudiantPasContentException(){
    System.out.println("Je suis exceptionnel");
  }
}

class Etudiant{
  public void recevoirNote(int n) throws
      EtudiantPasContentException{
    if(n < 10) throw new EtudiantPasContentException();
}

class Test{
  public static void main(String[] args){
    Etudiant e = new Etudiant();
    e.recevoirNote(3); // il ne va pas être content :)
}
```

Les différents constructeurs de la classe `Exception`

```
// Constructs a new  exception with null as its detail message.
Exception()

// Constructs a new exception with the specified detail message.
Exception(String message)

// Constructs a new exception with the specified detail message
    and cause.
Exception(String message, Throwable cause)

// Constructs a new exception with the specified detail
    message, cause, suppression enabled or disabled, and
    writable stack trace enabled or disabled.
protected Exception(String message, Throwable cause, boolean
    enableSuppression, boolean writableStackTrace)

// Constructs a new exception with the specified cause and a
    detail message of (cause==null ? null : cause.toString())
    (which typically contains the class and detail message of
    cause).
Exception(Throwable cause)
```

Table des matières

Gestion d'une pile

- Écrire une classe `Pile` qui implémente une pile d'objets avec un tableau de taille fixe avec comme attributs :

```
private final static int taille = 10;
private Object [] pile;
private int pos;
```

- On définira pour cela deux exceptions `PilePleine` et `pileVide`
- On utilisera pour écrire les méthodes, l'exception `ArrayOutOfBoundsException` qui indique qu'on a tenté d'accéder à une case non définie d'un tableau
- Écrire une méthode `main` qui empile les arguments de la ligne de commande (tant que c'est possible) et qui les réécrit dans l'ordre inverse

Correction : gestion d'une pile

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

class PileVide extends Exception{
  PileVide(String s) { super(s); }
}

class Test{
  public static void main(String[] args){
    Pile p = new Pile();

    try {
      for(int i=0;i<args.length;i++) p.empile(args[i]);
    }
    catch(PilePleine e) {};

    try {
      for(;;) System.out.print(p.depile()+" ");
    }
    catch(PileVide e) { System.out.println(); }
  }
}
```

Correction : gestion d'une pile

```
class Pile{
  private final static int taille = 10;
  private Object [] pile;
  private int pos;
  Pile() { pile=new Object[taille]; pos=0; }
  public void empile(Object o) throws PilePleine{
    try {
      pile[pos]=o;
      pos++;
    }
    catch(ArrayIndexOutOfBoundsException e){
      throw new PilePleine("Pile pleine!");
    }
  }
  public Object depile() throws PileVide{
    try {
      Object o = pile[pos-1];
      pos--;
      return o;
    }
    catch(ArrayIndexOutOfBoundsException  e){
      throw new PileVide("Pile vide!");
    }
  }
}
```