Programmation Orientée Objet: Java version 2021-22

Enseignante : Mme HAYET LAMINE ABID

Niveau: 2ème année Licence

A.U. 2021-2022

Programmation Orientée Objet: Java version 2021-22

Chapitre 5 : Les exceptions

Enseignante : Mme HAYET LAMINE ABID

Niveau: 2ème année Licence

A.U. 2021-2022

Plan

- Introduction au monde Objet
- La langage Java: Introduction et caractéristiques
- Syntaxe et structures de contrôle
- Héritage
- Les exceptions

Exception : Définition

- La gestion des erreurs dans un programme est une activité importante et souvent complexe dans les langages classiques

- Au lieu de gérer les erreurs à l'aide des constantes (comme en C), on peut créer dans nos programmes des classes supplémentaires d'exception

- Une exception est un **objet** particulier de la classe **`Exception`** ou d'une des ses sous classes qui est crée au moment où une situation anormale est détectée
- Les exceptions sont rencontrées dans de nombreuses situations

- Une exception est une sorte de signal qui indique qu'une erreur ou une situation anormale est intervenue
- Une méthode ayant détecté une situation anormale déclenche une exception. Cette exception sera capturée par le code

Exception : Exemple

- Certaines exécutions peuvent faire apparaître (lever) des exceptions

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        A cannot be resolved to a variable

        at Exemple1.main(Exemple1.java:7)
```

- utilisation des clauses **try** et **catch**

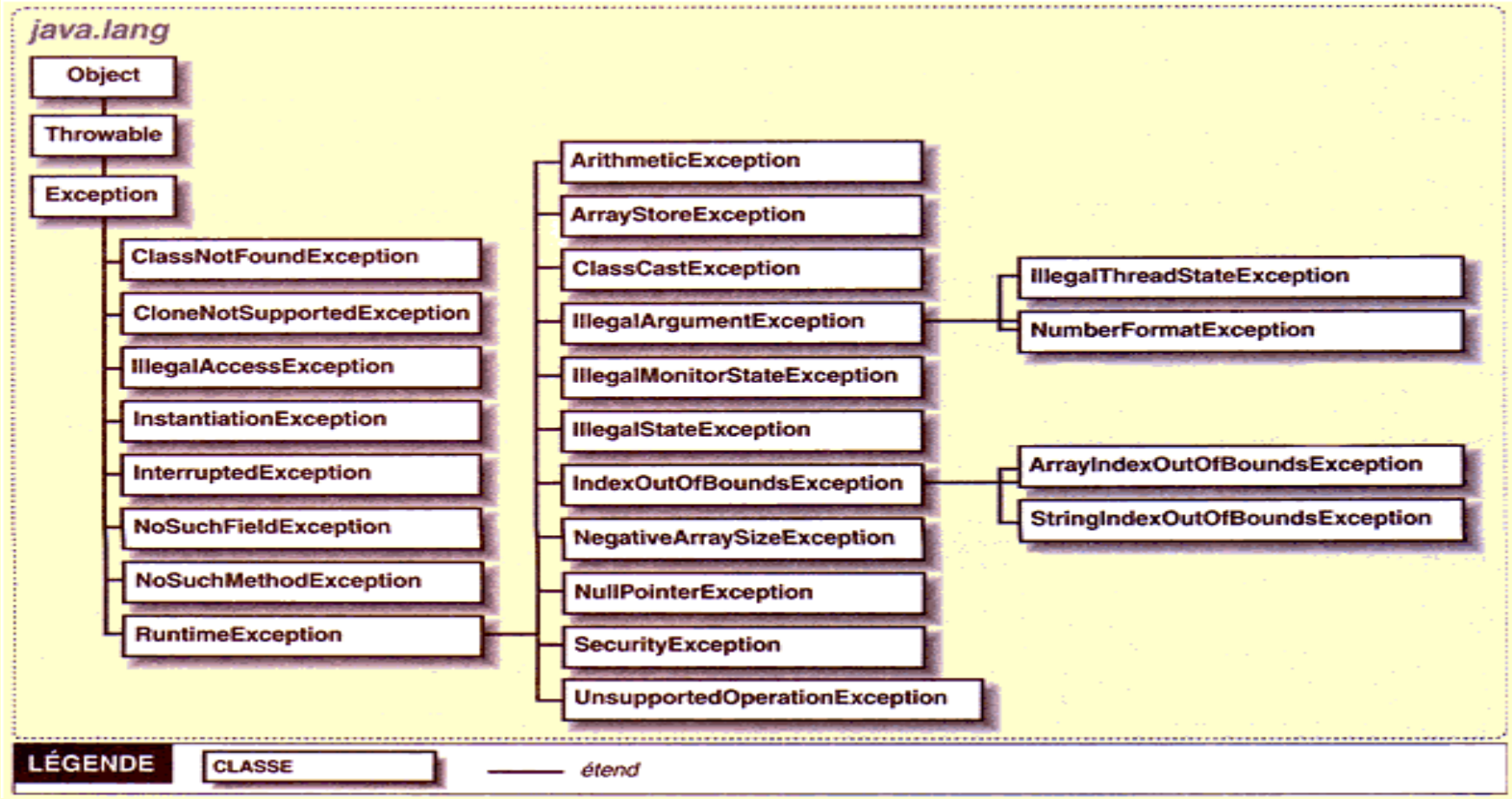
```
public class Exemple1 {
public static void main(String argv[])
{
try {
int val=Integer.parseInt(A[0]);
}
catch (NumberFormatException except) {
System.out.println("erreur");
return; // on sort sans créer d'instance
}
}}
```

Exception : Mécanisme

- Lorsqu'une situation exceptionnelle est rencontrée, une exception est lancée
- Si cette exception n'est pas attrapée dans le bloc où elle a été lancée, elle est propagée au niveau du bloc englobant,
- Si celui-ci ne l'attrape pas, elle est transmise au bloc de niveau supérieur, et ainsi de suite …
- Si l'exception n'est pas attrapée dans la méthode qui la lance, elle est propagée dans la méthode qui invoque cette dernière
- Si la structure de bloc de la méthode d'invocation ne contient aucune instruction attrapant l'exception, celle-ci est à nouveau propagée vers la méthode de niveau supérieur.
- Si une exception n'est jamais attrapée :
  - *Propagation jusqu'à la méthode main() à partir de laquelle l'exécution du programme a débutée,*
  - *Affichage d'un message d'erreur et de la trace de la pile des appels (call stack),*
  - *Arrêt de l'exécution du programme.*

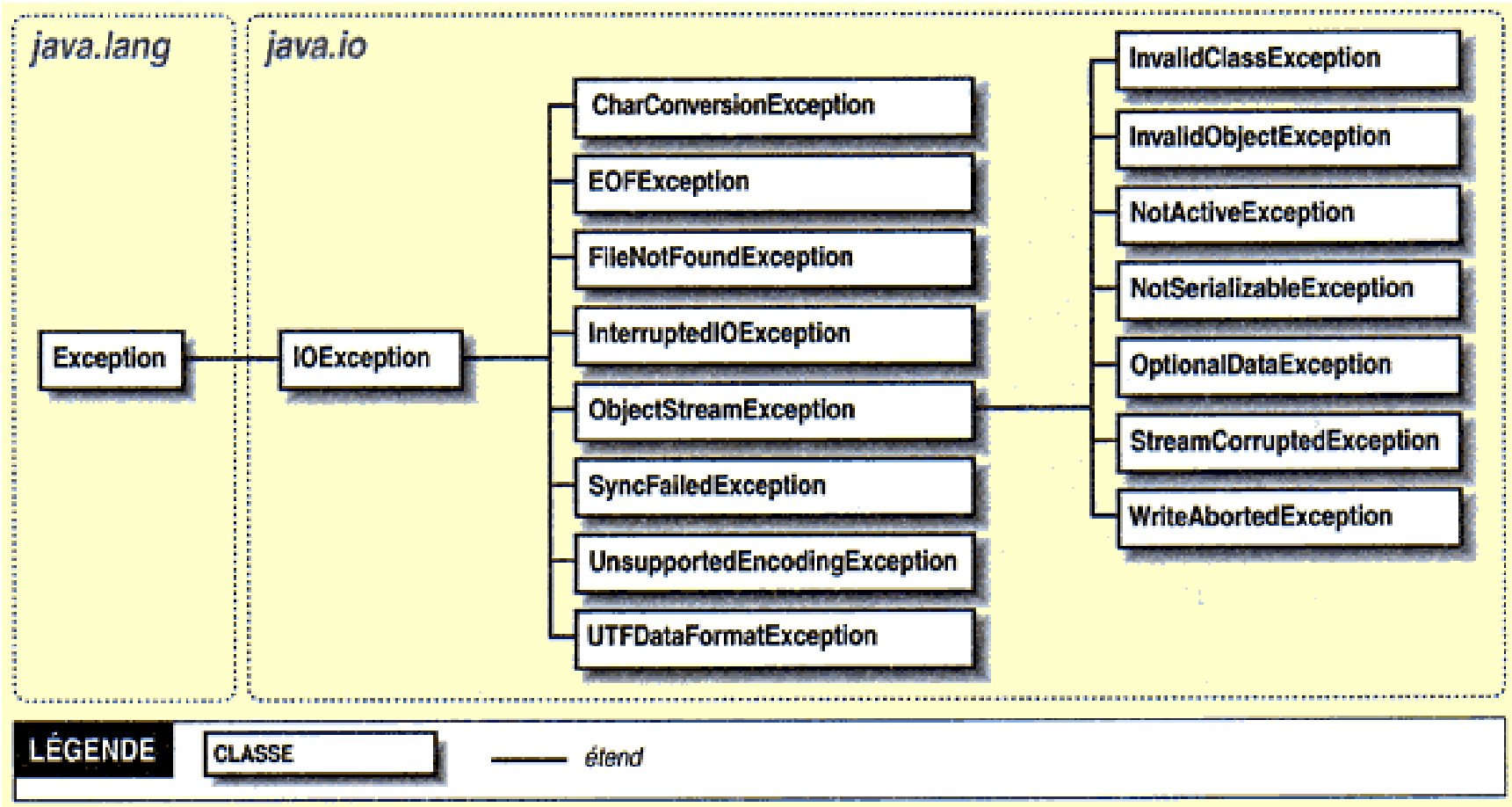
Exception : Objet Exception

- En Java, les exceptions sont des objets
  - *Toute exception doit être instance d'une sous-classe de la classe Java.lang.Throwable*



Exception : Objet Exception

- En Java, les exceptions sont des objets
  - *Toute exception doit être instance d'une sous-classe de la classe Java.lang.Throwable*



Exception : Traitement

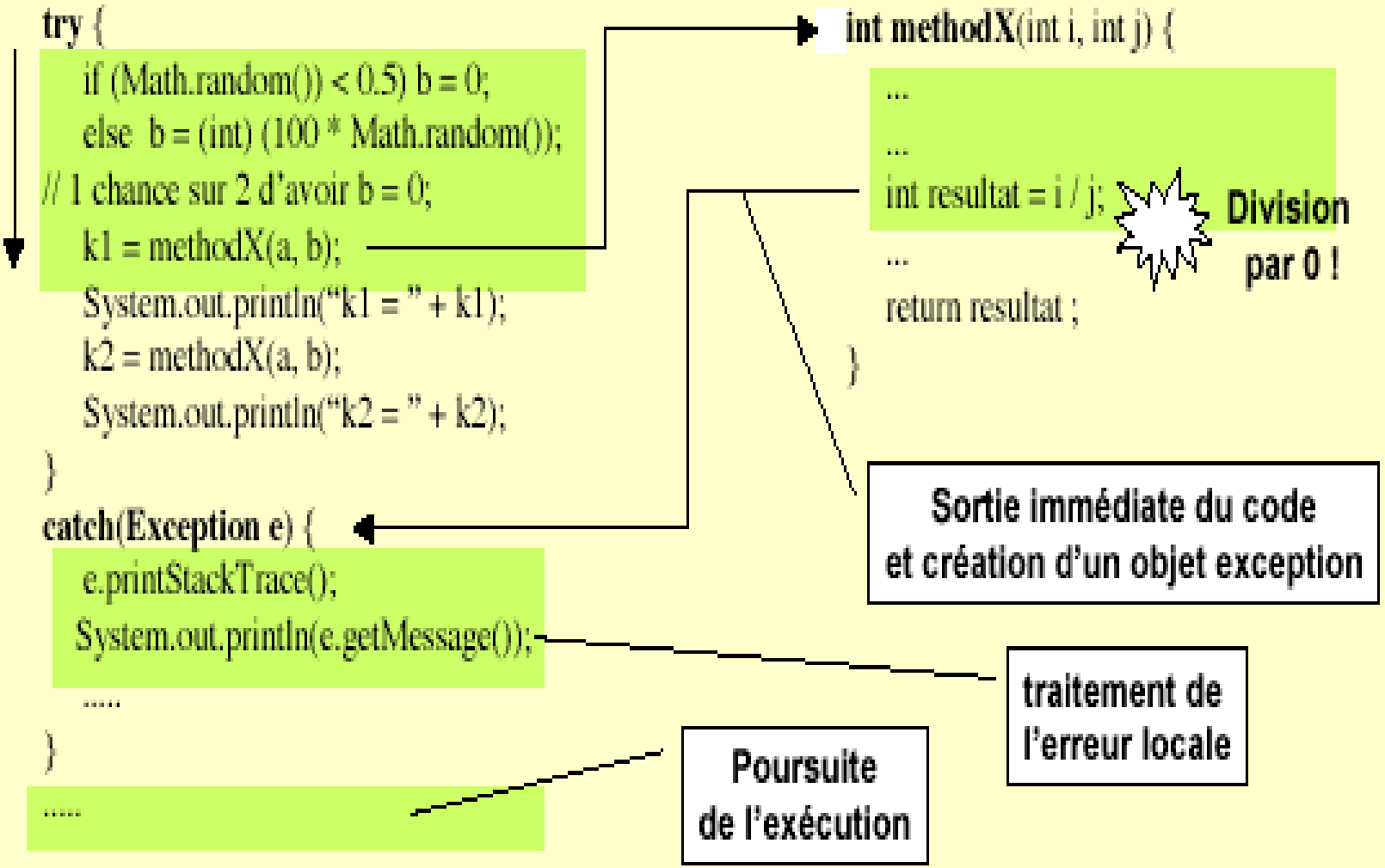
- **try** délimite un ensemble d'instructions susceptibles de déclencher une ou plusieurs types d'exceptions pour lesquelles une gestion est mise en oeuvre.

```
try{ ...
        // code pouvant lever des exceptions
 }
```

- **catch** : Cette gestion est réalisée par des blocs qui suivent le bloc try {...}. Les blocs catch interceptent (attrapent) les exceptions dont le type est spécifié afin d'exécuter du code spécifique à la gestion de l'erreur

```
catch(TypeDexception e)
{
        ...
}
```

Exception : Traitement



Exception : Traitement...

- Un bloc try peut-être suivi de plusieurs clauses catch qui permettent d'intercepter les exceptions dont le type est spécifié (dans la clause catch) et d'exécuter alors du code spécifique à l'exception.
- Un **seul bloc** catch **peut-être exécuté** : le premier susceptible d'attraper l'exception.
  - *Chaque clause catch doit être déclarée avec un argument de type Throwable ou une sous-classe de Throwable*
  - *Quand une exception est levée dans le bloc try, la première clause catch dont le type de l'argument correspond à celui de l'exception levée est invoquée*
    - *clause catch dont l'argument est de même classe que l'exception levée,*
    - *clause catch dont l'argument est une super-classe de la classe de l'exception levée.*

Exception : Traitement...

- L'ordre des blocs catch est donc important !

- Il n'est pas nécessaire d'avoir une clause catch pour chaque type d'exception pouvant être levée.

- si aucun bloc catch ne permet d'attraper l'exception, celle-ci est propagée vers la méthode appelante qui a alors la charge de la traiter ou non.

- Cela permet de traiter différentes types d'exceptions à différents endroits du programme.

- si une exception n'est attrapée par aucune des méthodes présentes dans la pile des appels, alors un message d'erreur ainsi que la trace de la pile d'appels sont affichés et l'exécution du programme est interrompue.

Exception : le bloc finally

- Le clauses catch sont suivies de manière optionnelle par un bloc **finally**
- Le bloc finally contient du code qui sera exécuté quelle que soit la manière dont le bloc try a été quitté.
- Le bloc finally permet de spécifier du code dont l'exécution est garantie quoi qu'il arrive :
  - *le bloc try s'exécute normalement sans qu'aucune exception ne soit levée*
    - *la fin du bloc try a été atteinte,*
    - *contrôle quitte le bloc try suite à une instruction return, continue, break*
  - *le bloc try lève une exception attrapée par l'un des blocs catch.*
  - *le bloc try lève une exception qui n'est attrapée par aucun des blocs catch qui le suivent.*

Exception : Traitement...

- Intérêt double :
  - *permet de rassembler dans un seul bloc un ensemble d'instructions qui autrement auraient pu être dupliquées*
  - *permet d'effectuer des traitements après le bloc try, même si une exception a été levée et non attrapée par les blocs catch*

Exception : Traitement...

```
...
try {
// ouvrir un fichier
// effectuer des traitements
// susceptibles de lever une
// exceptions
// fermer le fichier
...
}
catch (CertainesExceptions e)
{
// traiter l'exception
// Fermer le fichier
}
catch (AutresExceptions e) {
// traiter l'exception
// Fermer le fichier}
```

```
...
try {
// ouvrir un fichier
// effectuer des traitements
// susceptibles de lever une
// exceptions
// fermer le fichier
...
}
catch (CertainesExceptions e) {
// traiter l'exception
}
catch (AutresExceptions e) {
// traiter l'exception
}
finally {
// fermer le fichier
}
```

Exception : définition des exceptions utilisateur…

- On peut définir nos propres exceptions au moyen de l'instruction ***throw***

```
public class MyException extends Exception
 {
        …
}
```

- L'instruction throw(unObjetException) permet de lancer une exception
  - *unObjetException doit être une instance d'une sous-classe de Throwable*

```
throw new MyException();
```

  - Quand une exception est lancée,
  - *l'exécution du programme est interrompue,*
  - *la JVM recherche la clause catch la plus proche permettant de traiter l'exception lancée,*
  - *cette recherche se propage au travers des blocs englobants et remonte les appels de méthodes jusqu'à ce qu'un gestionnaire de l'exception soit trouvé (catch)*
  - *Tous les blocs finally rencontrés au cours de cette propagation sont exécutés.*

Exception : Traitement...

```
class
OperationBancaireException
extends Exception {
/**
* Compte pour lequel
l'opération a échouée
*/
private Compte c;
public
OperationBancaireException
(Compte c, String s) {
super(s);
This.c=c;
}
```

Si la pré-condition n'est pas vérifiée, création et lancement d'une exception

L'exception doit être déclarée dans la signature de la méthode

```
class Compte {
protected double solde = 0;
.....
/**


•Dépôt d'argent sur le compte
* s, la somme a déposer, s doit être >= 0
* throws OperationBancaireException si s <= 0
*/
public void créditer(double s) throws
OperationBancaireException
 {
if(s < 0) throw(new
OperationBancaireException (this, "dépôt
incorrect")) ;
solde += s;
}
}
```

L'opération ne peut être effectuée que si le dépôt est >= 0

Exception : Exemple

```
class ExceptionRien extends
Exception { public String
toString() { return("Aucune note
n'est valide"); } }
```

```
class ExceptionThrow {
static int moyenne(String[]
liste) throws ExceptionRien {
int somme=0,entier, nbNotes=0;
int i;
 for (i=0;i < liste.length;i++)
{
try {
entier=Integer.parseInt(liste[i]
);
 somme+=entier; nbNotes++; }
catch (NumberFormatException e)
{ System.out.println("La
"+(i+1)+" eme note n'est "+ "pas
entiere"); } }
 if (nbNotes==0) throw new
ExceptionRien();
return somme/nbNotes; }
```

la méthode **moyenne** indique ainsi qu'il est possible qu'une de ses instructions envoie une exception de type **ExceptionRien** sans que celle-ci soit attrapée par un mécanisme **try-catch**.

on demande ainsi le lancement d'une instance de **ExceptionRien**. Une fois lancée, l'exception se propagera. Ici, il y aura sortie de la méthode moyenne, on se retrouve alors à l'appel de la méthode moyenne dans le main, appel qui se trouve dans un "bloc **try**" suivi d'un "bloc **catch**" qui attrape les ExceptionRien(s) : l'instruction de ce "bloc **catch**" est effectuée. Le programme reprend alors son cours normal (pour se terminer).

```
public static void main (String[]
A) {try { System.out.println("La
moyenne est "+moyenne(A)); }

catch (ExceptionRien e)

{ System.out.println(e); } } }
```

Redéclenchement d'une exception

•Dans un gestionnaire d'exception, il est possible de demander que, malgré son traitement, l'exception soit retransmise à un niveau englobant, comme si elle n'avait pas été traitée. Il suffit pour cela de la relancer en appelant à nouveau l'instruction throw.

```
class Point {
   public Point(int x, int y) throws ErrConst {
      if ( (x<0) || (y<0)) throw new ErrConst ( ) ;
      this.x = x ; this.y = y ;
   }
   public void f ( ) throws ErrConst {
      try {
         Point p = new Point (-3, 2);
      }
      catch (ErrConst erreur)  {
         System.out.println ( "dans catch (ErrConst) de f" ) ;
         throw erreur; // on repasse l'exception à un niveau supérieur
      }
   }
   private int x, y;
}

class ErrConst extends Exception { ... }

public class Except7 {
   public static void main(String args[]) {
      try {
         Point a = new Point(1, 4);
         a.f ( );
      }
      catch (ErrConst erreur)  {
         System.out.println ( "dans catch (ErrConst) de main" ) ;
      }
      System.out.println ( "après bloc try main" ) ;
   }
}
```

Exception : Conclusion

- Toute méthode susceptible de lever une exception doit :
  - *Soit l'attraper,*
  - *Soit la déclarer explicitement, c'est-à-dire comporter dans sa signature l'indication que l'exception peut être provoquée : clause throws*
- Les exceptions déclarées dans la clause throws d'une méthode sont:
  - *les exceptions levées dans la méthode et non attrapées par celle-ci,*
  - *les exceptions levées dans des méthodes appelées par la méthode et non attrapées par celle-ci.*
- si on oublie de traiter, une exception, une erreur de compilation se déclenche