

# Les exceptions

# Introduction

- Les exceptions servent à gérer les erreurs qui peuvent arriver dans un programme.
- Connexion à une base de données échouée.
- Erreur de programmation : une méthode appliquée sur une référence nulle.
- Pas de gestion automatique d'exceptions : obligation de débrouiller avec code d'erreur.

# Les exceptions : les outils

- Attraper les exceptions : `try, catch, finally`
- Lancer des exceptions : `throw`
- Signaler les exceptions non attrapées : `throws`

# ArrayIndexOutOfBoundsException

```
public class TestException {  
    public static void main (String [] args) {  
        int[] t=new int [5]; // indices de 0 à 4  
        System.out.println("début ");  
        int x= t[5]; // arrêt du programme  
        System.out.println("affectation réussie!");  
    }  
}
```

Une exception non attrapée provoque l'arrêt du programme

# ArithmeticException

```
public class TestException {  
    public static void main (String [] args) {  
        int[] t={6,2,7,9,7};  
        System.out.println("début ");  
        int x= 1/(t[4]-t[2]) ; // arrêt du programme  
        System.out.println("affectation réussie!");  
    }  
}
```

Ici aussi, l'exception non attrapée provoque l'arrêt du programme

# Lancer, attraper une exception

En cas d'anomalie( index en dehors des limites, division par zéro...),  
une exception est lancée

Une exception lancée et n'est pas attrapée  
provoque l'arrêt du programme  
avec un message d'erreur

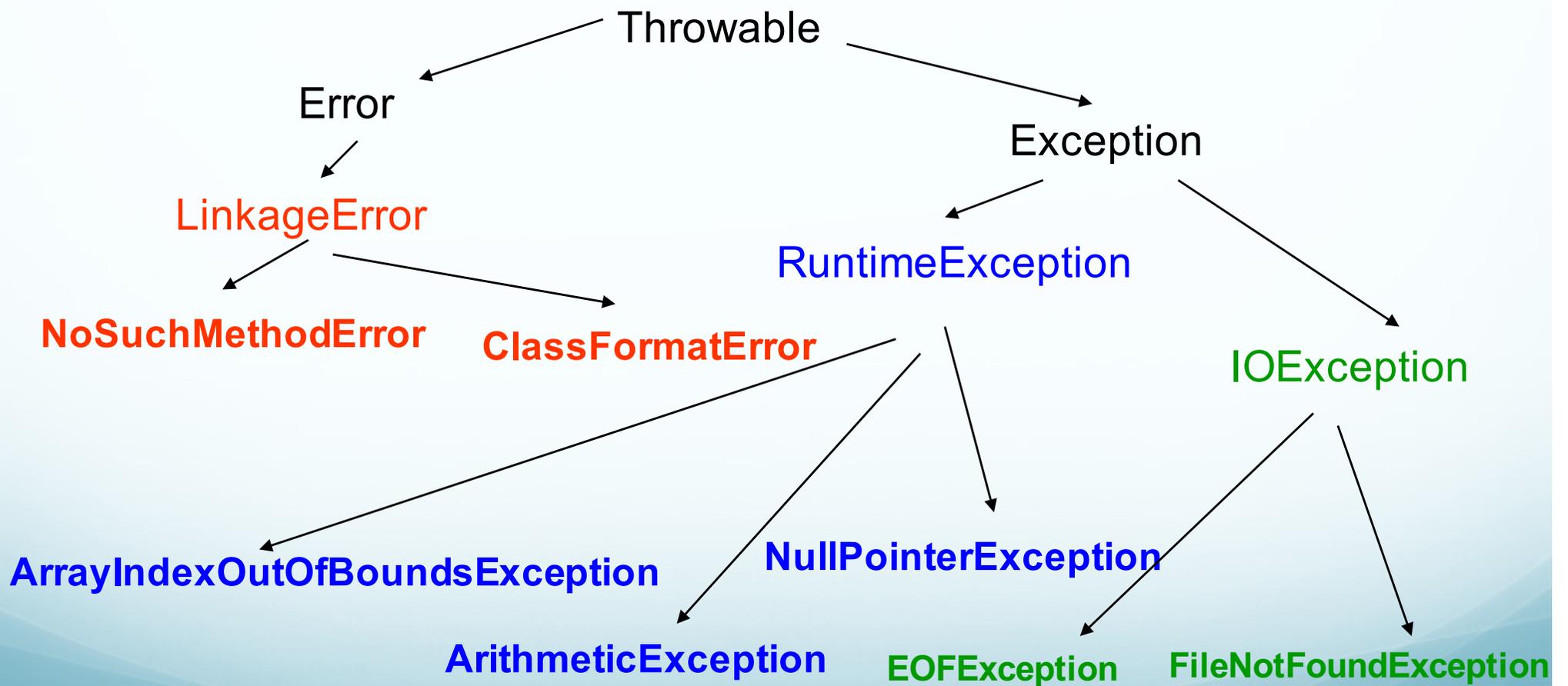
Pour gérer les exceptions,  
On utilise des instructions JAVA spécifiques

**MOT-CLES** liés aux exceptions:  
try, catch, finally, throw, throws

# Classe d'exceptions

- Une exception est une instance d'une classe d'exception
- La classe `java.lang.Throwable` est la classe racine des classes d'exceptions.
- Quelques classes d'exceptions :
  - `ArithmeticException` (pour des erreurs comme la division par zéro)
  - `ArrayIndexOutOfBoundsException` (indice  $<0$  ou  $\geq$  au nombre d'éléments du tableau, du vecteur...)
  - `NumberFormatException` (une chaîne qui n'a pu être convertie en nombre)
  - `FileNotFoundException` (fichier non trouvé)

# Quelques classes prédéfinies d'exceptions



# catch : pour attraper les exceptions

```
public class TestException {  
  
    public static void main (String [] args) {  
  
        int[] t={6,2,7,9,7};  
  
        try {  
  
            System.out.println("début ");  
  
            Int x= 1/(t[4]-t[2]) ;  
  
            System.out.println("affectation reussie!");  
  
            catch (ArithmeticException e) {  
  
                System.out.println ( "Operation impossible " ); }  
  
                System.out.println("Fin") ;  
  
            } }  
    }
```

# catch multiples

```
public class TestException {  
  
    public static void main (String [] args) {  
  
        int[] t={6,2,7,9,7};  
  
        try {  
  
            System.out.println("début ");  
  
            Int x= 1/(t[4]-t[2]) ; Int y=t [5];  
  
            System.out.println("affectation reussie!");  
  
            catch (ArithmeticException e) {System.out.println ( "Operation impossible " ); }  
  
            Catch(ArrayIndexOutOfBoundsException e ) {System.out.println ( "Operation  
                impossible " ); }  
  
            System.out.println("Fin"); } }  
}
```

# Où capturer les exceptions?

```
public class TestException {
```

```
public static void affectation( int i, int [] tab) {
```

```
    Int x=1/(tab[i+2] -tab[i] ); int y =tab[i+4] ;}
```

les exceptions sont souvent capturées

```
public static void main (String [] args) {
```

dans les fonctions appelantes

```
int[] t={6,2,7,9,7};
```

```
try {
```

```
System.out.println("début ");
```

```
affectation(2,t);
```

```
System.out.println("affectation reussie!");}
```

```
catch (ArithmeticException e) {System.out.println ( "Operation impossible " ); }
```

```
catch(ArrayIndexOutOfBoundsException e ) {System.out.println ( "Indice incorrect " ); }
```

```
System.out.println("Fin" ); }
```

# Où capturer les exceptions?

```
public class TestException {  
  
    public static void affectation( int i, int [] tab) {  
  
        try {  
  
            Int x=1/(tab[i+2] -tab[i] ); int y =tab[i+4] ;  
  
            catch (ArithmeticException e) {System.out.println ( "Operation impossible " ); }  
  
            System.out.println("Fin de effectuer ");  
  
        }  
  
        public static void main (String [] args) {  
  
            int[] t={6,2,7,9,7};  
  
            try {  
  
                System.out.println("début ");  
  
                affectation(2,t);  
  
                System.out.println("affectation reussie!");}  
  
            catch(ArrayIndexOutOfBoundsException e ) {System.out.println ( "indice incorrect " ); }  
  
            System.out.println(Fin) ; } }
```

Le contrôle est transmis au  
premier bloc catch de la bonne classe

# Finally: à exécuter quoiqu' il arrive?

```
public class TestException {  
  
    public static void affectation( int i, int [] tab) {  
  
        try {  
  
            int x=1/(tab[i+2] -tab[i] ); int y =tab[i+4] ;  
  
            catch (ArithmeticException e) {System.out.println ( "Operation impossible " );}  
  
            Finally   {}{System.out.println ( "Fin de effectuer" );}  
  
            System.out.println("Fin de effectuer "); }  
  
        public static void main (String [] args) {  
  
            int[] t={6,2,7,9,7};  
  
            try {  
  
                System.out.println("début");  
  
                affectation(2,t); System.out.println("affectations terminees!");}  
  
                catch(ArrayIndexOutOfBoundsException e ) {System.out.println ( "indice incorrect " ); }  
  
                System.out.println(Fin) ; } }
```

# Finally: illustration

```
• public class TestException {
  public static void main (String [] args) {
    int[] tab={6,2,7,9,7};
    try{
      try {
        System.out.println("début ");
        int x=1/(tab[4]-tab[2]);
        int y=tab[6];
        System.out.println("affectations terminees!");
      }catch(ArrayIndexOutOfBoundsException e ){
        System.out.println ("indice incorrect "); }
      finally {
        System.out.println ("Fin de affectuer"); }
    }catch (ArithmeticException e) {
      System.out.println ("Operation impossible "); }

    System.out.println("Fin");
  } }
```

- Execution  
Début  
Fin de affectuer  
Operation impossible  
Fin

# Comment déclencher une exception avec `throw` nous même

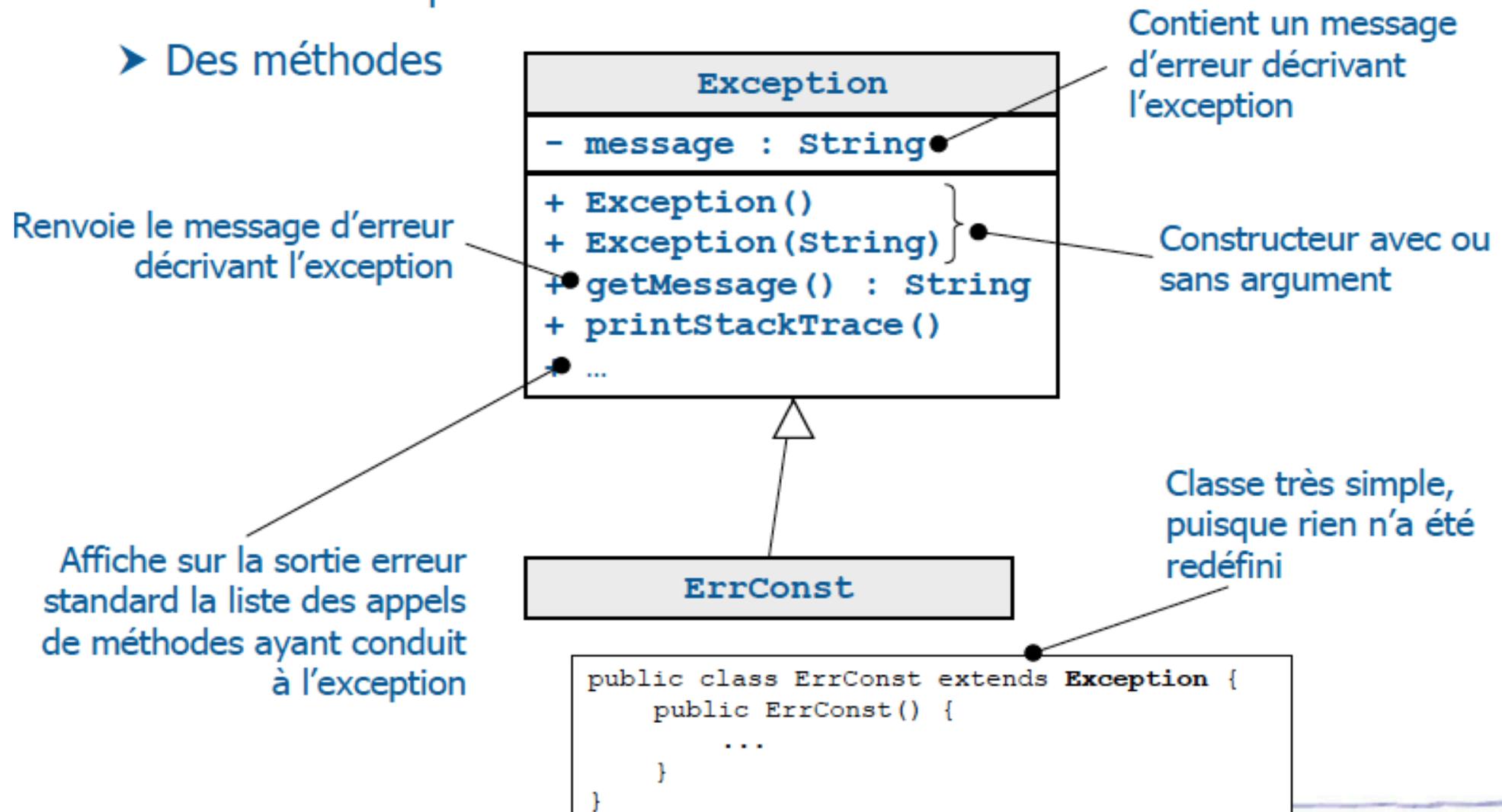
- Considérons une classe `Point`, munie d'un `constructeur` à deux arguments et d'une `méthode affiche`
- Supposons que nous manipulons que des points ayant des coordonnées non négatives.
- Nous pouvons, au sein du constructeur, vérifier la validité des paramètres fournis.
- Lorsque l'un d'entre eux est incorrect, nous "`déclenchons`" (on emploie aussi les verbes "`lancer`" ou "`lever`") une exception à l'aide de l'instruction `throw`.

# Suite (2)

- Nous créons donc (un peu artificiellement) une classe que nous nommerons `ErrConst`
- Java impose que cette classe dérive de la classe standard `Exception`:  
`class ErrConst extends Exception { };`
- Pour lancer une exception de ce type au sein de notre constructeur, nous fournirons à l'instruction `throw` un objet de type `ErrConst`, par exemple de cette façon :
- `throw new ErrConst();` (*Objet anonyme*)

# Suite (3)

- Les exceptions sont des objets nous pouvons donc définir
  - Des attributs particuliers
  - Des méthodes



# Suite(3)

En définitive, le constructeur de notre classe Point peut se présenter ainsi :

```
class Point {
    public Point(int x, int y) throws ErrConst {
        if ( (x<0) || (y<0) ) throw new ErrConst(); // lance une
        this.x = x; this.y = y; // exception de type ErrConst
    }
    public void affiche() { System.out.println("<" + x + ", " + y + ">"); }
    private int x, y;
}

class ErrConst extends Exception {}
```

Dans l'en-tête du constructeur, **throws ErrConst**, précise que la méthode est susceptible de déclencher une exception de type **ErrConst**. Cette indication est obligatoire en Java, à partir du moment où l'exception en question n'est pas traitée par la méthode elle-même.

# Utilisation d'un gestionnaire d'exception

- Comment procéder pour gérer convenablement les éventuelles exceptions de type `ErrConst` que son emploi peut déclencher.
- Pour ce faire, il faut :
  - 1- inclure dans un bloc particulier dit "`bloc try`" les instructions dans lesquelles on risque de voir déclenchée une telle exception ;
  - 2- faire suivre ce bloc de la définition des différents gestionnaires d'exception (ici, un seul suffit). Chaque définition de gestionnaire est précédée d'un en-tête introduit par le mot clé `catch` (comme si `catch` était le nom d'une méthode gestionnaire).

# Suite (1)

```
public class Except1 {  
    public static void main(String args[]) {  
        try {  
            Point a = new Point(1, 4);  
            a.affiche();  
            a = new Point(-3, 5);  
            a.affiche();  
        }  
        catch (ErrConst e) {  
            System.out.println ("Erreur construction ");  
            System.exit (-1);  
        }  
    }  
}
```

Résultats obtenus

---

<1, 4>  
Erreur construction

Si une exception est déclenchée par une instruction situé dans le bloc try, alors :

- 1-Le programme abandonne l'exécution du code restant dans le bloc try.
- 2-Le programme exécute le code du gestionnaire situé dans la clause catch.

# Suite(2)

- Si aucune des instructions du bloc `try` ne lance une exception, le programme ignore la clause `catch`.
- Ce premier exemple est très restrictif pour différentes raisons :
  - 1- on n'y déclenche et on ne traite qu'un seul type d'exception ;
  - 2- le gestionnaire d'exception ne reçoit aucune information ; plus exactement, il reçoit un objet sans valeur qu'il ne cherche pas à utiliser ;
  - 3- nous n'exploitons pas les fonctionnalités de la classe `Exception` dont dérive notre classe `ErrConst` ;
  - 4- le gestionnaire d'exception se contente d'interrompre le programme alors qu'il est possible de poursuivre l'exécution.

# Gestion de plusieurs exceptions

Nous considérons une classe Point munie :

- du constructeur précédent, déclenchant toujours une exception **ErrConst**,
- d'une méthode **déplace** qui s'assure que le déplacement ne conduit pas à une coordonnée négative ; si tel est le cas, elle déclenche une exception **ErrDepl** (on crée donc, ici encore,

```
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 déplace(int dx, int dy) throws ErrDepl {  
        if ( [ x+dx<0 ] || [ y+dy<0 ] ) throw new ErrDepl();  
        x += dx; y += dy;  
    }  
    public void affiche() { System.out.println( "<x+> <y+>" ); }  
    private int x, y;  
}
```

```
class ErrConst extends Exception { }  
class ErrDepl extends Exception { }
```

# Gestion de plusieurs exceptions

```
public class Except2 {  
    public static void main(String args[]) {  
        try {  
            Point a = new Point(1, 4);  
            a.affiche();  
            a.déplace(-3, 5);  
            a = new Point(-3, 5);  
            a.affiche();  
        }  
        catch (ErrConst e) {  
            System.out.println ("Erreur construction ");  
            System.exit (-1);  
        }  
        catch (ErrDepl e) {  
            System.out.println ("Erreur déplacement ");  
            System.exit (-1);  
        }  
    }  
}
```

Résultats obtenus

<1, 4>

Erreur déplacement

Nous nous contentons comme précédemment d'afficher un message et d'interrompre l'exécution

# Transmission d'information au gestionnaire d'exception

On peut transmettre une information au gestionnaire d'exception :

- 1- par le biais de l'objet fourni dans l'instruction **throw**,
- 2- par l'intermédiaire du constructeur de la classe **Exception**.

# Par l'objet fourni à l'instruction **throw**

```
class Point {
    public Point(int x, int y) throws ErrConst {
        if ( (x<0) || (y<0) ) throw new ErrConst(x, y);
        this.x = x; this.y = y;
    }
    public void affiche() { System.out.println("<" + x + ", " + y + ">"); }
    private int x, y;
}

class ErrConst extends Exception {
    ErrConst(int abs, int ord) { this.abs = abs; this.ord = ord; }
    public int abs, ord;
}

public class Except3 {
    public static void main(String args[]) {
        try {
            Point a = new Point (1, 4);
            a.affiche();
            a = new Point (-3, 5);
            a.affiche();
        }
        catch (ErrConst e) {
            System.out.println ("Erreur construction Point");
            System.out.println ("Coordonnées souhaitées : " + e.abs + ", " + e.ord);
            System.exit (-1);
        }
    }
}
```

## Résultats obtenus

<1, 4>

Erreur construction **Point**

Coordonnées souhaitées : -3, 5

# Par le constructeur de la classe Exception

```
class Point {
    public Point(int x, int y) throws ErrConst {
        if ( [x<0] || [y<0])
            throw new ErrConst("Erreur construction coord : "+ x +", "+ y);
        this.x = x; this.y = y;
    }
    public void affiche() { System.out.println("<" + x +", "+ y + ">"); }
    private int x, y;
}

class ErrConst extends Exception {
    ErrConst(String message) { super(message); }
}

public class Except4 {
    public static void main(String args[]) {
        try {
            Point a = new Point (1, 4);
            a.affiche();
            a = new Point (-3, 5);
            a.affiche();
        }
        catch (ErrConst e) {
            System.out.println ( e.getMessage() );
            System.exit (-1);
        }
    }
}
```

## Résultats obtenus

<1, 4>

Erreur construction coord : -3, 5

# Poursuite de l'exécution

```
public class Except5 {  
    public static void main(String args[]) {  
        System.out.println("Avant bloc try");  
        try {  
            Point a = new Point (1, 4);  
            a.affiche();  
            a.déplace (-3, 5);  
            a.affiche();  
        }  
        catch (ErrConst e) { System.out.println ("Erreur construction"); }  
        catch (ErrDepl e) { System.out.println ("Erreur déplacement"); }  
        System.out.println("Après bloc try");  
    }  
}
```

## Résultats obtenus

---

Avant bloc try  
<1, 4>  
Erreur déplacement  
Après bloc try

# Choix du gestionnaire d'exception

```
class ErrPoint extends Exception { ... }  
class ErrConst extends ErrPoint { ... }  
class ErrDepl extends ErrPoint { ... }
```

```
void f () throws ErrConst, ErrDepl {  
    ...  
    throw new ErrConst ();  
    ...  
    throw new ErrDepl ();  
    ...  
}
```

```
try {  
    ... // on suppose qu'on utilise f  
}  
catch (ErrPoint e) {  
    ... // on traite ici à la fois les exceptions de  
    ... // type ErrConst et celles de type ErrDepl  
}
```

```
try {  
    ... // on suppose qu'on utilise f  
}  
catch (ErrConst e) {  
    ... // on traite ici uniquement les exceptions de type ErrConst  
}  
catch (ErrDepl e) {  
    ... // on traite ici uniquement les exceptions de type ErrDepl  
}
```

# Cheminement des exceptions et la clause `throws`

- Lorsqu'une méthode déclenche une exception, on cherche tout d'abord un gestionnaire dans l'éventuel bloc `try` contenant l'instruction `throw` correspondant.
- Si l'on n'en trouve pas ou si aucun bloc `try` n'est prévu à ce niveau, on poursuit la recherche dans un éventuel bloc `try` associé à l'instruction d'appel dans une méthode appelante, et ainsi de suite.
- Toute méthode susceptible de déclencher une exception qu'elle ne traite pas localement doit mentionner son type dans une clause `throws` figurant dans son en-tête. Lorsqu'il existe plusieurs exceptions, il faut toutes les mentionner en les séparant par l'opérateur virgule.

# Cheminement des exceptions et la clause `throws`

```
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 déplace(int dx, int dy) throws ErrDepl {
        if ((x+dx)<0 || (y+dy)<0) throw new ErrDepl( );
        x += dx; y += dy;
    }
    public void affiche( ) { System.out.println("<" + x + ", " + y + ">"); }
    private int x, y;
}

class ErrPoint extends Exception { ... }
class ErrConst extends ErrPoint { ... }
class ErrDepl extends ErrPoint { ... }

public class Except6 {
    public static void main(String args[]) {
        try {
            TestPoint( );
        }
        catch (ErrPoint e) { System.out.println ( "Erreur sur un point" ) ; }
    }
    static void TestPoint( ) throws ErrConst, ErrDepl {
        Point a = new Point(1, 4);
        a.affiche( ); // cette méthode ne capture pas les exceptions et se
        a.déplace(-3, 5); // de les propager grâce à throws, c'est la méthode
        a.affiche( ); // appelante main qui s'en occupe
    }
}
```

# 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`.
- Cette possibilité de re-déclenchement d'une exception s'avère très précieuse lorsque l'on ne peut résoudre localement qu'une partie du problème posé.

# Redéclenchement d'une exception

```
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");
    }
}
```

## Résultats obtenus

dans catch (ErrConst) de f  
dans catch (ErrConst) de main  
après bloc try main

# Le bloc `finally`

- Java permet d'introduire, à la suite d'un bloc `try`, un bloc particulier d'instructions qui seront toujours exécutées :
- soit après la fin "naturelle" du bloc `try`, si aucune exception n'a été déclenchée .
- soit après le gestionnaire d'exception (à condition, bien sûr, que ce dernier n'ait pas provoqué d'arrêt de l'exécution).
- Ce bloc est introduit par le mot clé `finally` et doit obligatoirement être placé après le dernier gestionnaire. Voici un exemple de code, accompagné de deux exemples d'exécutions :

# Le bloc finally

```
class Point {
    public Point(int x, int y) throws ErrConst {
        if ( (x<0) || (y<0) ) throw new ErrConst ();
        this.x = x; this.y = y;
    }
    private int x, y;
}

class ErrConst extends Exception { ... }

public class Except8 {
    public static void main(String args[]) {
        try {
            System.out.println ("début bloc try main");
            int n = saisie.Clavier.lireInt ("donner un entier : ");
            Point a = new Point(n, n);
            System.out.println("fin bloc try main");
        }
        catch (ErrConst erreur) {
            System.out.println ("dans catch (ErrConst) de main" );
        }
        finally {
            System.out.println("exécution du bloc finally");
        }
        System.out.println ("après bloc try main" );
    }
}
```

## Résultats obtenus

---

début bloc **try** main  
donner un entier : 5  
fin bloc **try** main  
exécution du bloc **finally**  
après bloc **try** main

## Résultats obtenus

---

début bloc **try** main  
donner un entier : -5  
dans **catch** (ErrConst) de main  
exécution du bloc **finally**  
après bloc **try** main

# Exemple

un exemple de programme qui détecte les exceptions standard `NegativeArraySizeException` et `ArrayIndexOutOfBoundsException` et qui utilise la méthode `getMessage` pour afficher le message correspondant

```
import saisie.Clavier;

public class Except9 {
    public static void main(String[] args) {
        try {
            int tableau[];
            int n = Clavier.lireInt("Taille voulue : ");
            tableau = new int[n];
            int i = Clavier.lireInt("indice : ");
            tableau[i] = 12;
            System.out.println("***fin normale");
        }
        catch (NegativeArraySizeException e) {
            System.out.println("Taille négative : "+ e.getMessage());
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("indice tableau : "+ e.getMessage());
        }
    }
}
```

Résultats obtenus

---

Taille voulue : -2  
Taille négative : -2

Résultats obtenus

---

Taille voulue : 10  
indice : 15  
indice tableau : 15

# Comparaison: Exception ou pas

**Exemple** : gérer les erreurs sans les exceptions

```
erreurType lireFichier() {
    int codeErreur = 0;
    // Ouvrir le fichier
    if (isFileIsOpen()) {
        // Détermine la longueur du fichier
        if (getFileSize()) {
            // Vérification de l'allocation de la mémoire
            if (getEnoughMemory()) {
                // Lire le fichier en mémoire
                if (readFailed()) {
                    codeErreur = -1;
                }
            } else {
                codeErreur = -2;
            }
        } else {
            codeErreur = -3;
        }

        // Fermeture du fichier
        if (closeTheFileFailed()) {
            codeErreur = - 4;
        }
    } else {
        codeErreur = - 5;
    }
}
```

# Avec les exceptions:

```
void lireFichier() {
    try {
        // Ouvrir le fichier
        // Détermine la longueur du fichier
        // Vérification de l'allocation de la mémoire
        // Lire le fichier en mémoire
        // Fermer le fichier
    } catch (FileOpenFailed) {
        ...
    } catch (FileSizeFailed) {
        ...
    } catch (MemoryAllocFailed) {
        ...
    } catch (FileReadFailed) {
        ...
    } catch (FileCloseFailed) {
        ...
    }
}
```