

**R2-01-03**

**DÉVELOPPEMENT ORIENTÉ OBJETS**

**QUALITÉ DE DÉVELOPPEMENT**

---

## **Semaine 4**

- Valeur et référence
- Iterator
- Exception

**Francis Brunet-Manquat et Hervé Blanchon**

Université Grenoble Alpes

IUT 2 – Département Informatique

# Points abordés

---


 Exercice 1 : suppression d'un élément dans une liste

 Wrapper et iterator

 Exercice 2 : table de multiplication

 Gestion des interactions

 Exception

 Exercice « fil rouge » : la bataille de Faërun

# VALEUR ET RÉFÉRENCE EN JAVA

# Que contient une variable ?

---

 Le contenu d'une variable est différent selon que ...

 ... c'est une variable de **type primitif**

 ... c'est une variable de **type Classe** (un objet)


 Il est important de faire la différence...

# Les types primitifs...

## Booléen

 **boolean** (valeurs true et false)


## Caractère

 **char** (unicode de \u0000 à \uffff, 128 premiers : codes ASCII)

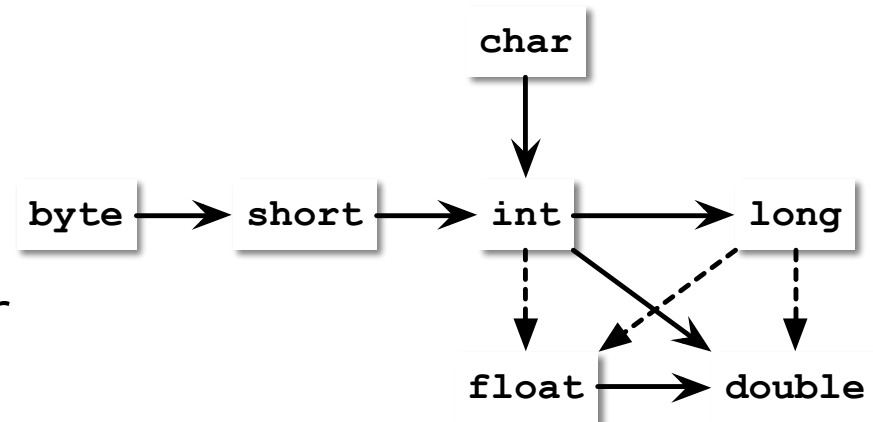
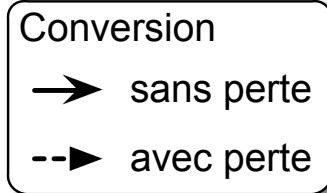
## Entier (... taille représentation)

 **byte** (1 octet), **short** (2 octets), **int** (4 octets), **long** (8 octets)

## Réel (... taille représentation)

 **float** (4 octets), **double** (8 octets)

➔ *Conversions possibles avec l'opérateur **cast***



 **(nouveau\_type) valeur**

# Contenu d'une variable...

## ... de Type primitif

 déclaration-initialisation

```
int i = 12;
```

 état de la mémoire

variable	adresse	contenu	mémoire
i	\$ff34ef24	12	

 À RETENIR !

 **une variable de type primitif  
contient une valeur**

## ... de Type Classe d'Objet


 déclaration-initialisation

```
String s = new String("ABC");
```

 état de la mémoire

variable	adresse	contenu	mémoire
s	\$ff34ef68	\$ff34effa	
	\$ff34effa	"ABC"	

 À RETENIR !

 **une variable de type Classe  
contient une référence** à un objet  
(un pointeur vers un objet,  
l'adresse d'un objet)

# CLASSES ENVELOPPES

## *WRAPPERS*

# Pourquoi ? À quoi ça sert ?

 Quand on a besoin d'un **Objet « de type primitif »**

 On utilise la classe enveloppe (wrapper) qui lui correspond (elle encapsule les données du type primitif)

Type primitif	Classe enveloppe	
boolean	<b>B</b> oolean	
char	<b>C</b> har <b>a</b> cter	
byte	<b>B</b> yte	sous-classes de la classe Number
short	<b>S</b> hort	
int	<b>I</b> nteger	
long	<b>L</b> ong	
float	<b>F</b> loat	
double	<b>D</b> ouble	

 On utilise une instance de classe enveloppe comme une instance d'objet ordinaire



# La documentation

---


## **Boolean**

 [http://docs.oracle.com/javase/7/docs/api/java/lang/\*\*Boolean\*\*.html](http://docs.oracle.com/javase/7/docs/api/java/lang/Boolean.html)

## **Character**

 [http://docs.oracle.com/javase/7/docs/api/java/lang/\*\*Character\*\*.html](http://docs.oracle.com/javase/7/docs/api/java/lang/Character.html)


## **ClasseY**

 [http://docs.oracle.com/javase/7/docs/api/java/lang/\*\*ClasseY\*\*.html](http://docs.oracle.com/javase/7/docs/api/java/lang/<b>ClasseY</b>.html)

# NOTION D'ITÉRATEUR & UTILISATION

# Définitions

---

 Iterator est une interface définie dans le package `java.util`


```
public interface Iterator {  
    boolean hasNext();  
    Object next();  
    void remove(); // Optional  
}
```

 `Iterator iterator()`

 renvoie un objet « itérateur » qui permet le parcours séquentiel des éléments d'une collection

 pas de garantie concernant l'ordre dans lequel les éléments sont retournés

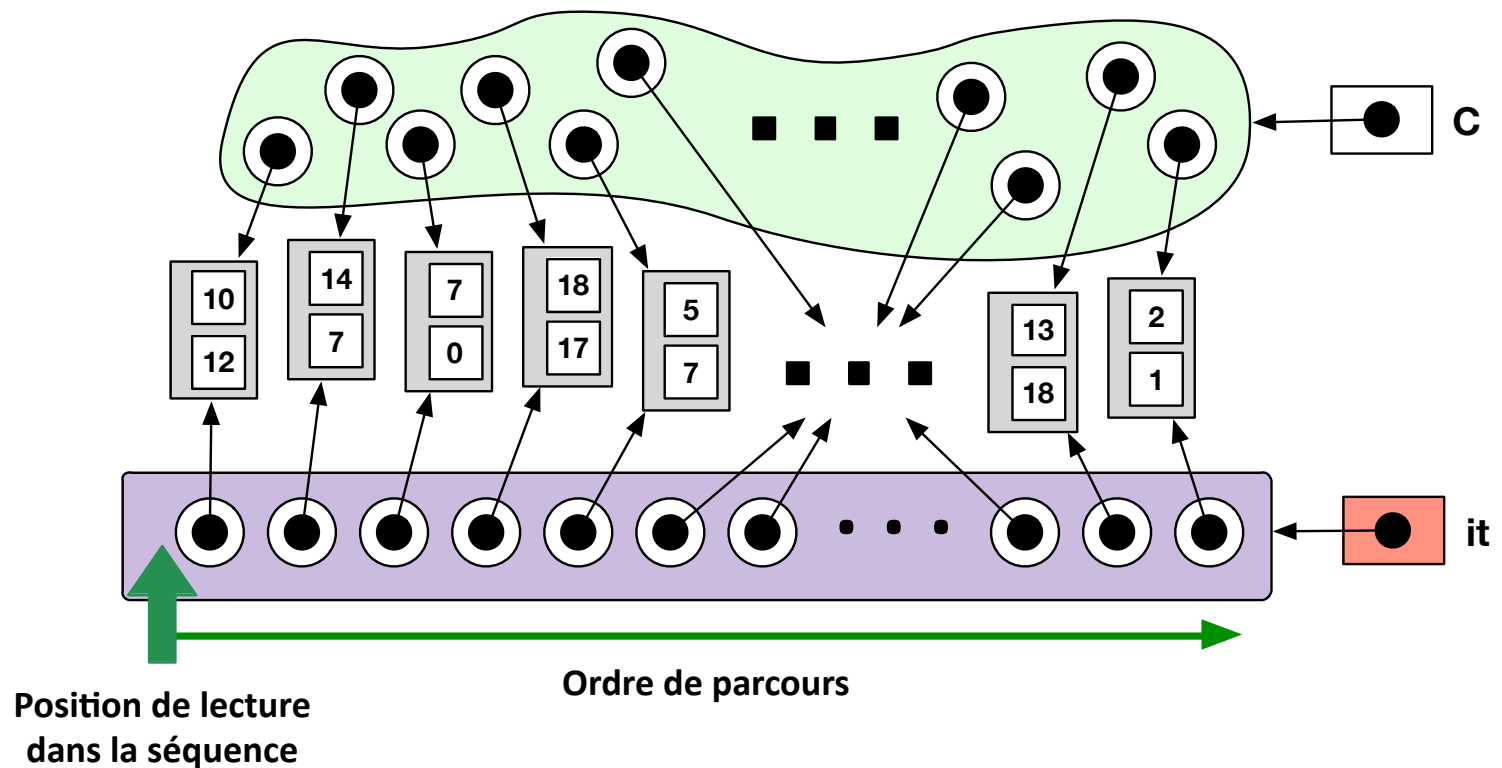
# Itérateur : création & initialisation

 `Iterator it = c.iterator();`

 crée `it` itérateur sur la Collection

 ordre de parcours

 position actuelle dans le parcours

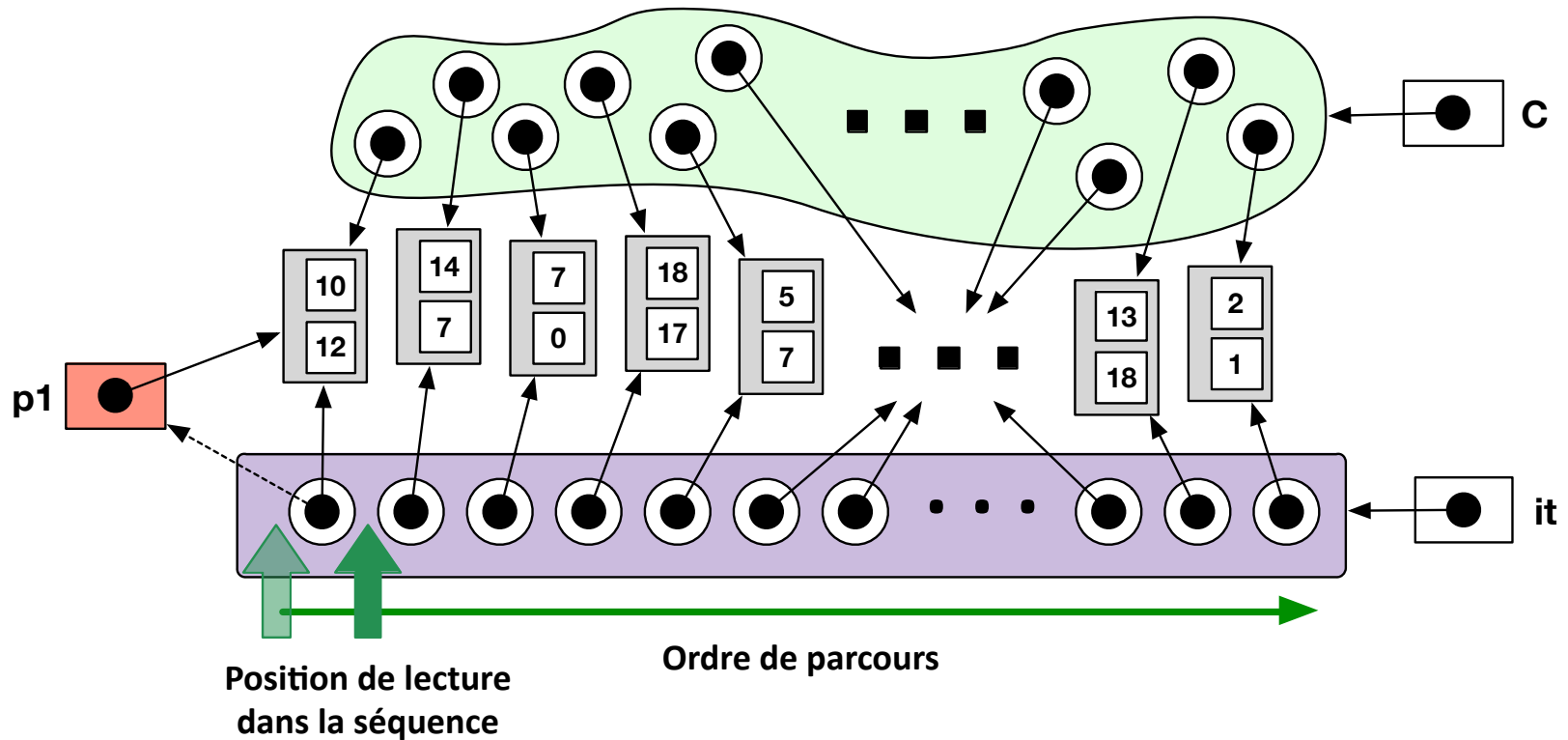


# Itérateur : accès au suivant

Point `p1 = it.next()`;

permet de récupérer la référence de l'élément suivant

`it` avance



# Itérateur : suivant ?

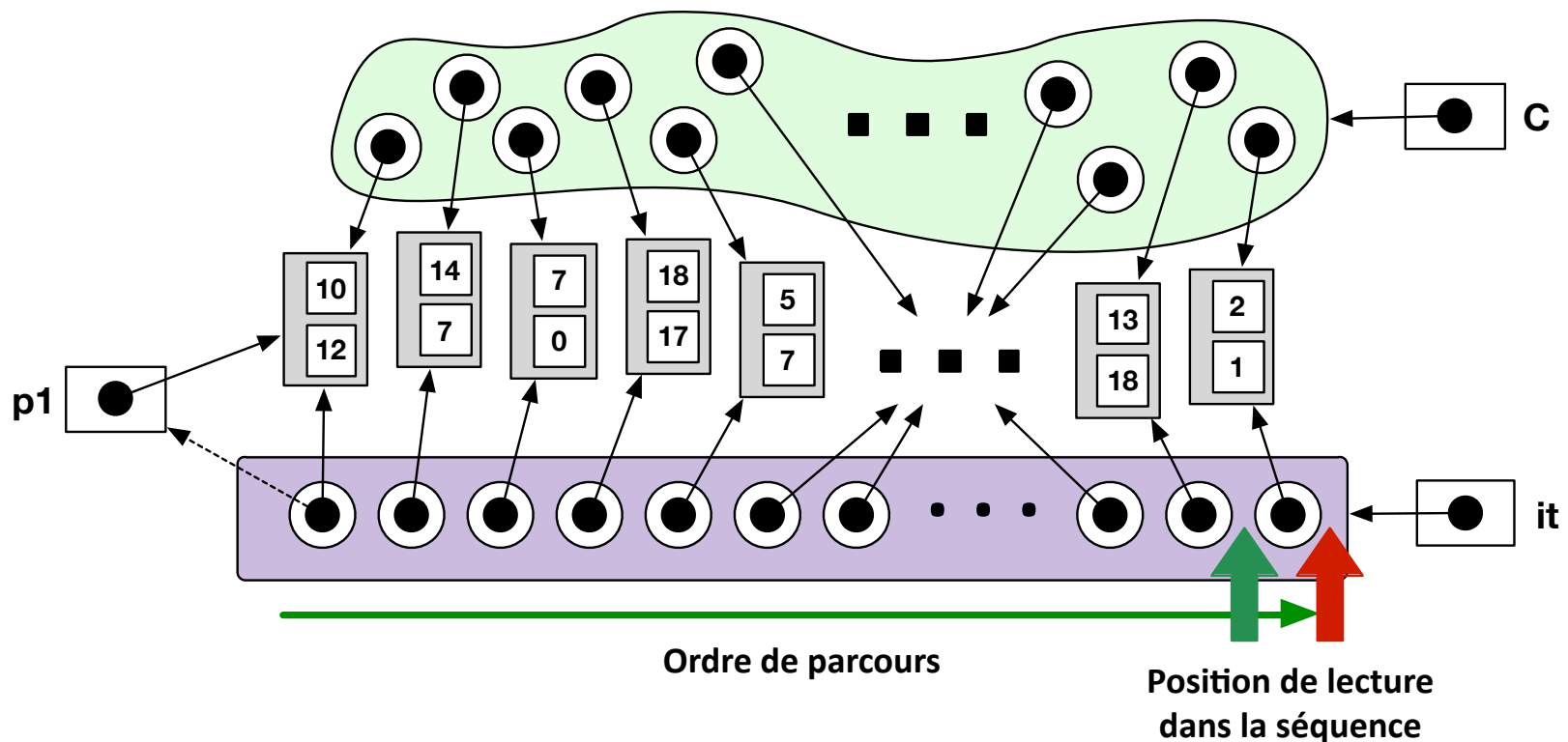
it.hasNext() // retourne un booléen

permet savoir si on a atteint la fin du parcours

it.hasNext() // retourne true

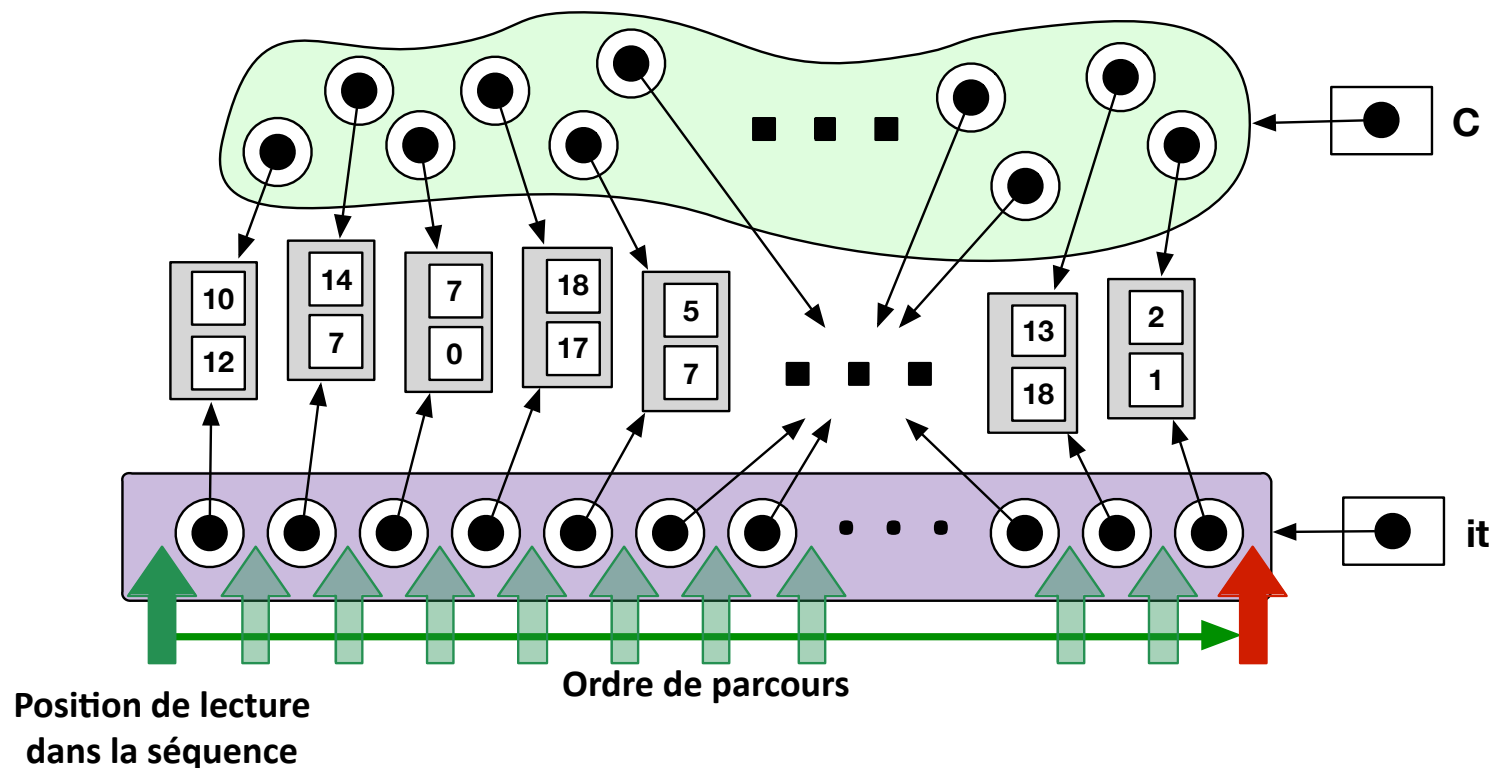
it.Next()

it.hasNext() // retourne false




# Itérateur : parcours complet

```
Iterator it = c.iterator();  
while (it.hasNext()) {  
    Point p = it.next();  
    // faire qq chose avec p  
}
```



# Itérateur : affichage d'une collection

---

 Pour afficher tous les éléments d'une collection on peut faire ainsi :

```
static void displayAll(Collection col) {  
    Iterator itr = col.iterator();  
    while (itr.hasNext()) {  
        String str = (String) itr.next();  
        System.out.print(str + " ");  
    }  
    System.out.println();  
}
```

 voir par exemple



[http://www.tutorialspoint.com/javaexamples/collection\\_all.htm](http://www.tutorialspoint.com/javaexamples/collection_all.htm)



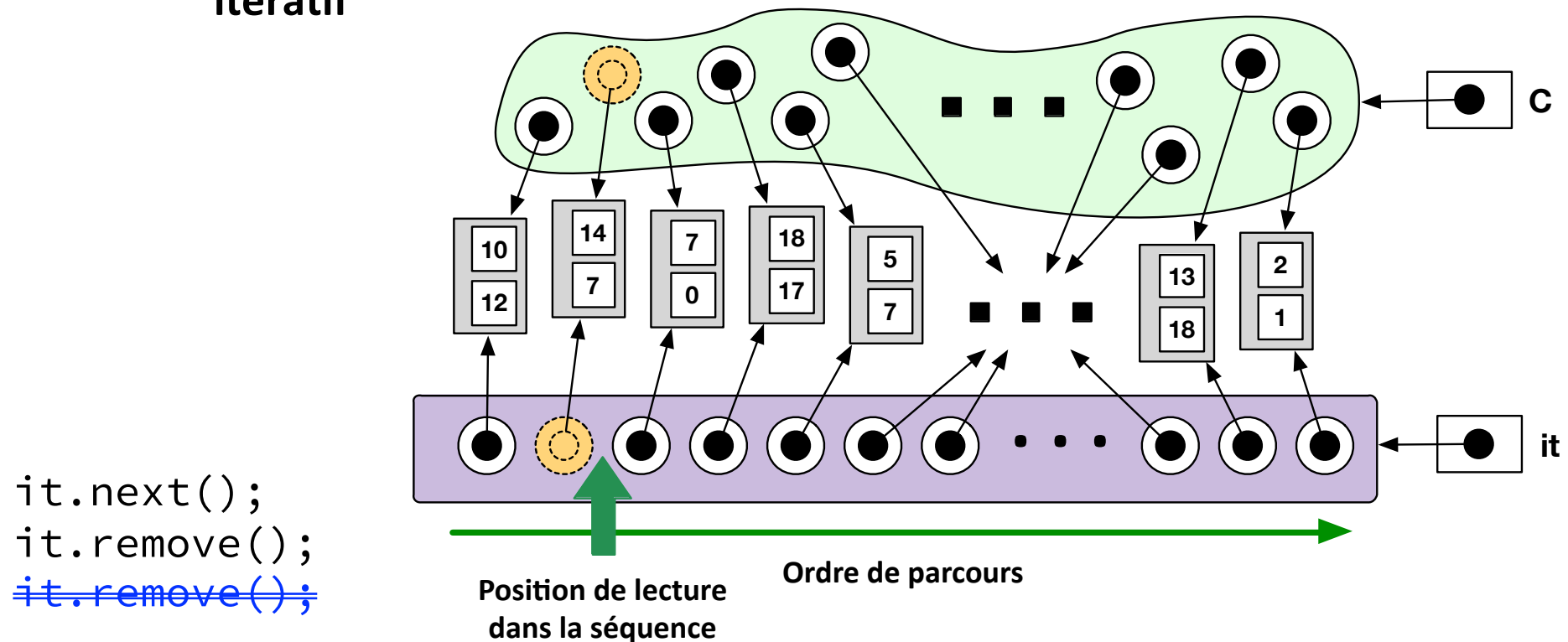
# Itérateur : suppression

it.remove();

retire de la collection le dernier élément « lu » (celui qui précède la position de lecture)

ne peut être invoqué d'une seule fois par appel de next()

seule manière sûre de modifier une collection dans un parcours itératif



# Itérateur : filtrage d'une Collection

---



Exemple :

```
static void filter(Collection c) {  
    for (Iterator i = c.iterator(); i.hasNext(); ) {  
        if (!cond(i.next()))  
            i.remove();  
    }  
}
```

# EXCEPTIONS EN JAVA

# Introduction

---

 Un programme peut rencontrer une erreur ou un événement anormal

 Erreur technique

 *Fichier non présent, plus de mémoire, etc.*

 Erreur métier

 *Arguments invalides, etc.*



 Prévoir un **traitement d'erreur** sur les instructions susceptibles de les provoquer

 En Java, ce traitement est intégré dans le langage : traitement des **exceptions**




# Exception : principe

---

 Une exception est créée :

-  soit par la JVM [environnement d'exécution] (erreur interne/technique)
-  soit par une levée d'exception du programmeur

 Deux solutions

-  **attraper** (**intercepter**) l'exception immédiatement **pour** la **traiter**
-  **relancer** (**propager**) l'exception à la méthode qui a déclenché le traitement erroné
  -  MAIS on devra **attraper** et **traiter** au cours de la **propagation**

# Attraper une exception : syntaxe de base

---

```
try {  
    ... // instructions à contrôler  
  
} catch (MonException e) {  
    ... // traitement de l'exception e  
  
} [catch (AutreException e) {...}]
```

# try-catch : exemple

Code	<pre>1 public class TestSimple { 2     public static void main(String args[]) { 3         int num1, num2; 4         try { 5             // bloc try pour circonscrire l'interception 6             num1 = 0; 7             num2 = 62 / num1; // division par zéro 8             System.out.println("On va quitter un bloc try sans problème"); 9         } catch (ArithmeticException e) { 10             // bloc pour intercepter une division par zéro 11             System.out.println("Erreur : il est interdit de diviser par 0!"); 12         } 13         System.out.println("Sortie du bloc try-catch"); 14         System.out.println("pour une suite d'exécution normale"); 15     } 16 }</pre>
Trace	<pre>7 // levée d'une exception 9 // interception de l'exception 11 Erreur : il est interdit de diviser par 0 ! 13 Sortie du bloc try-catch 14 pour une suite d'exécution normale</pre>

# Attraper une exception : finally

---

```
try {  
    ... // instructions à contrôler  
  
} catch (MonException e) {  
    ... // traitement de l'exception e  
[] catch (AutreException e) {...]  
} finally {  
    ... // instructions exécutées  
        // dans tous les cas  
}
```










# try-catch-finally : exemple 1

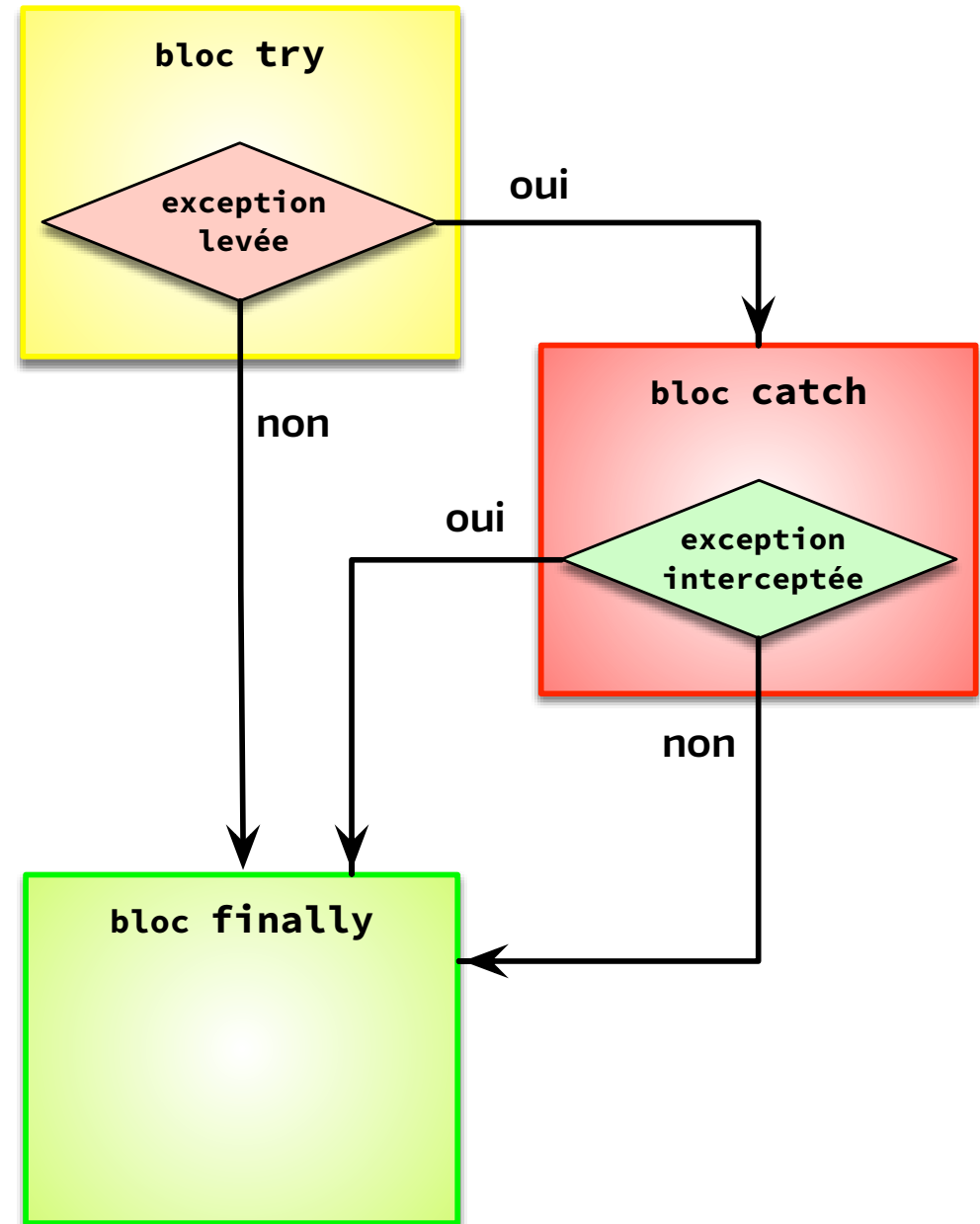
Code	<pre>1 public class TestSimple2 { 2     public static void main(String args[]) { 3         int num1, num2; 4         try { 5             // bloc try pour circonscrire l'interception 6             num1 = 0; 7             num2 = 62 / num1; // division par zéro 8             System.out.println("Try -&gt; bloc quitté sans problème"); 9         } catch (ArithmeticException e) { 10            // bloc pour intercepter une division par zéro 11            System.out.println("Catch -&gt; Erreur : division par 0!"); 12        } finally { 13            // bloc exécuté quoi qu'il arrive dans try et catch 14            System.out.println("Finally -&gt; exécuté tout le temps !"); 15        } 16        System.out.println("Sortie de la séquence try-catch-finally"); 17        System.out.println("pour une suite d'exécution normale"); 18    } 19 }</pre>
Trace	<pre>11 Catch -&gt; Erreur : division par 0! 14 Finally -&gt; exécuté tout le temps ! 16 Sortie de la séquence try-catch-finally 17 pour une suite d'exécution normale</pre>

# try-catch-finally : exemple 2

Code	<pre>1 public class TestSimple2 { 2     public static void main(String args[]) { 3         int num1, num2; 4         try { 5             // bloc try pour circonscrire l'interception 6             num1 = 1; 7             num2 = 62 / 1; // division par zéro 8             System.out.println("Try -&gt; bloc quitté sans problème"); 9         } catch (ArithmeticException e) { 10            // bloc pour intercepter une division par zéro 11            System.out.println("Catch -&gt; Erreur : division par 0!"); 12        } finally { 13            // bloc exécuté quoi qu'il arrive dans try et catch 14            System.out.println("Finally -&gt; exécuté tout le temps !"); 15        } 16        System.out.println("Sortie de la séquence try-catch-finally"); 17        System.out.println("pour une suite d'exécution normale"); 18    } 19 }</pre>
Trace	<pre>8 Try -&gt; bloc quitté sans problème 14 Finally -&gt; exécuté tout le temps ! 16 Sortie de la séquence try-catch-finally 17 pour une suite d'exécution normale</pre>

# Gestion du contrôle

-  une exception est levée dans le bloc **try**
-  le contrôle est transféré au bloc **catch**
-  lorsque le bloc **catch** s'est exécuté le contrôle est transféré au bloc **finally**
-  pas d'exception levée dans le bloc **try**
-  le contrôle est transféré au bloc **finally**
-  si une instruction **return** est présente dans les blocs **try** ou **catch**
-  le bloc **finally** est exécuté avant l'instruction **return**



# try-catch-finally : exemple 3

Code	<pre>1 public class TestSimpleAvecReturn { 2     private static int rendre0() { 3         try { 4             System.out.println(" rendre0 -&gt; début try"); 5             return 0; 6         } finally { 7             System.out.println(" rendre0 -&gt; finally exécuté avant return 0;"); 8         } 9     } 10    public static void main(String args[]) { 11        int val; 12        System.out.println("main -&gt; début"); 13        val = rendre0(); 14        System.out.println(" Résultat de rendre0 : " + val); 15        System.out.println("main -&gt; fin"); 16    } 17 }</pre>
Trace	<pre>12 main -&gt; début 4  rendre0 -&gt; début try 7  rendre0 -&gt; finally exécuté avant return 0; 13 // affectation val &lt;- 0 (suite au return) 14 Résultat de rendre0 : 0 15 main -&gt; fin</pre>

# Lever une exception : syntaxe

---

 `throw new Exception(...);`

 `Exception` est une classe définie par Java

 `throw new RuntimeException(...);`

 exceptions définies par Java

 `IOException`, `ParseException`, etc.

 exceptions définies dans le code (par le programmeur)

 héritant de la classe `Exception`

# (re)lancer une exception : syntaxe

---

- 🖥️ Ajouter dans la signature de la méthode qui relance le mot-clef **throws** suivi du type d'exception

```
public static int division(int c, int d)
    throws ArithmeticException {
    ...
}
```

- 🖥️ **ATTENTION** : il faut normalement attraper l'exception et la traiter à un moment de la « remontée »

## Lancer une exception : ArithmeticException

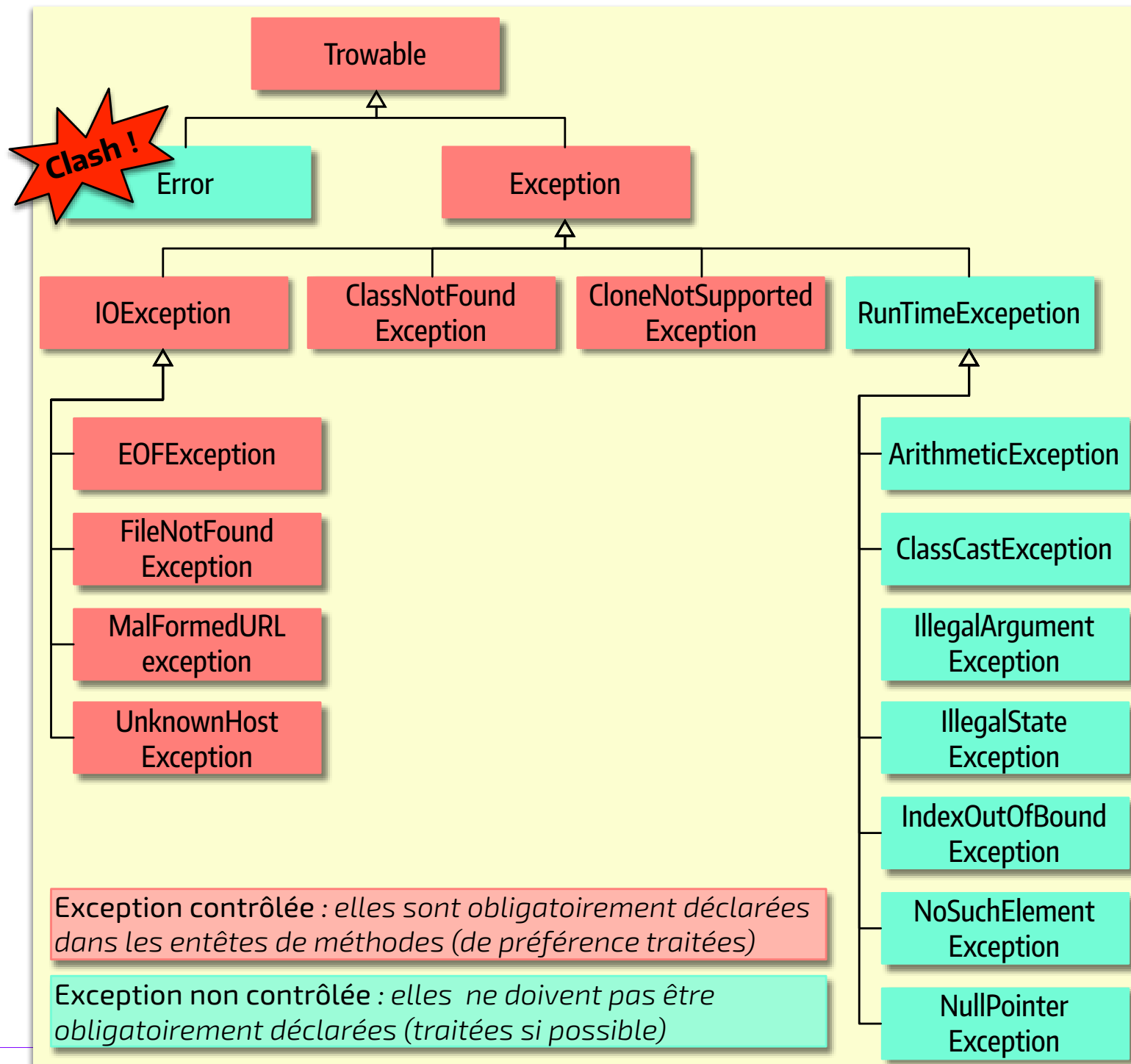
Code	<pre>12 public class TestThrow1 { 13 14     static void valider(int age) throws ArithmeticException { 15         if (age &lt; 18) { 16             throw new ArithmeticException("âge invalide"); 17         } else { 18             System.out.println("vous pouvez voter !"); 19         } 20     } 21 22     public static void main(String args[]) { 23         System.out.println("début du code..."); 24         valider(13); 25         System.out.println("suite du code..."); 26     } 27 28 }</pre>
Trace	<pre>23 début du code... Exception in thread "main" java.lang.ArithmeticException: âge invalide     at ExceptionPlayGround.TestThrow1.valider(<a href="#">TestThrow1.java:16</a>)     at ExceptionPlayGround.TestThrow1.main(<a href="#">TestThrow1.java:24</a>) Java Result: 1</pre>

## Lancer une exception : Exception

Code	<pre>12 public class TestThrow1 { 13 14     static void valider(int age) throws Exception { 15         if (age &lt; 18) { 16             throw new Exception("âge invalide"); 17         } else { 18             System.out.println("vous pouvez voter !"); 19         } 20     } 21 22     public static void main(String args[]) throws Exception { 23         // Exception doit obligatoirement être relancée 24         System.out.println("début du code..."); 25         valider(13); 26         System.out.println("suite du code..."); 27     } 28 }</pre>
Trace	<pre>23 début du code... Exception in thread "main" java.lang.Exception: âge invalide     at ExceptionPlayGround.TestThrow1.valider(<a href="#">TestThrow1.java:16</a>)     at ExceptionPlayGround.TestThrow1.main(<a href="#">TestThrow1.java:24</a>) Java Result: 1</pre>



# Hiérarchie des exceptions (extrait)



# Exception utilisateur MonException

---

 Les exceptions utilisateurs héritent de la classe Exception de Java

 elles ont un constructeur

 appel de super() si rien de particulier à faire

 elles ont des attributs si nécessaire

*cf. exemple sur le transparent suivant...*

```

class MonException extends Exception {
    // attribut privé
    private int ex;
    // constructeur
    MonException(int a) {
        ex = a;
    }
    // pour print
    public String toString() {
        return "MonException : " + ex + " < 0";
    }
}

```

```

class TestMonException {

    public static void main(String[] args) {
        try {
            ClasseUtilitaire.somme(-10, 10);
        } catch (MonException e) {
            System.out.println(e);
        }
    }
}

```

```

class ClasseUtilitaire {

    static void somme(int a, int b) throws MonException {
        if (a < 0) {
            throw new MyException(a);
        } else {
            System.out.println(a + b);
        }
    }
}

```

```

// dans main de TestMonException
// somme lève une exception : -10 < 0
// interceptée par catch pour afficher l'exception
MonException : -10 < 0

```

# Propagation d'une exception

```
class Classe2 {  
  
    Classe1 objet1 = new Classe1();  
    ...  
  
    void methodeX() {  
        try {  
            objet1.methode1();  
            ...  
        } catch (Exception exception1) {  
            // traitement exception  
            ...  
        } finally {  
            ...  
        }  
    }  
  
    void methodeY() throws Exception {  
        objet1.methode1();  
        ...  
    }  
}
```

```
class Classe1 {  
  
    ...  
  
    void methode1() throws Exception {  
        boolean erreur = false;  
        ...  
        erreur = ...;  
        // en cas d'erreur lever  
        // une exception Exception  
        if (erreur) {  
            throw new Exception();  
        }  
    }  
}
```

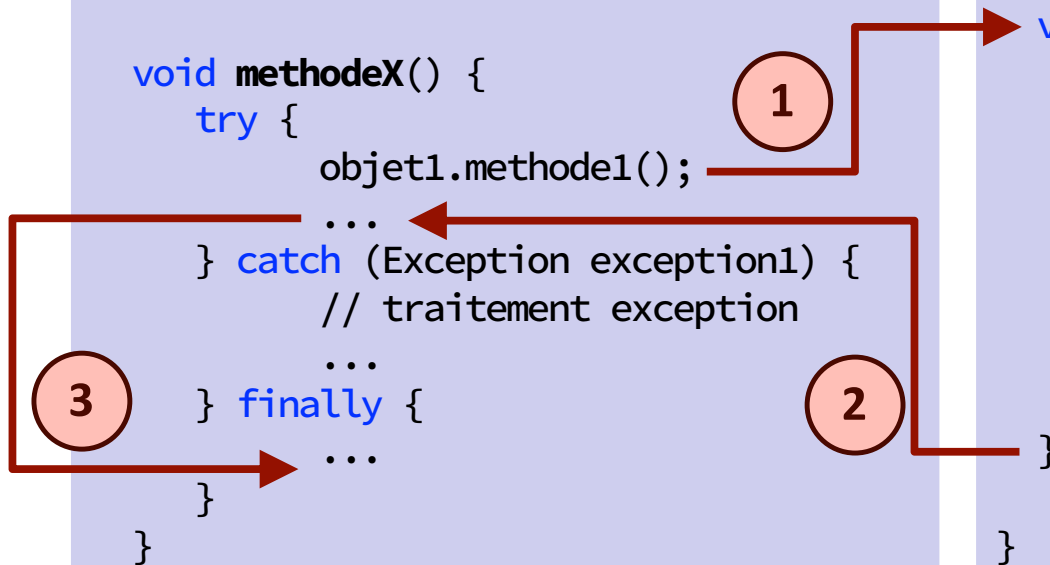


Classes fil rouge

# Propagation d'une exception

```
class Classe2 {  
    Classe1 objet1 = new Classe1();  
    ...  
    void methodeX() {  
        try {  
            objet1.methode1();  
            ...  
        } catch (Exception exception1) {  
            // traitement exception  
            ...  
        } finally {  
            ...  
        }  
    }  
    void methodeY() throws Exception {  
        objet1.methode1();  
        ...  
    }  
}
```

```
class Classe1 {  
    ...  
    void methode1() throws Exception {  
        boolean erreur = false;  
        ...  
        erreur = ...;  
        // en cas d'erreur lever  
        // une exception Exception  
        if (erreur) {  
            throw new Exception();  
        }  
    }  
}
```



## Exécution normale



`methodeX()` appelle `objet1.methode1()`



`methode1()` se déroule sans problème

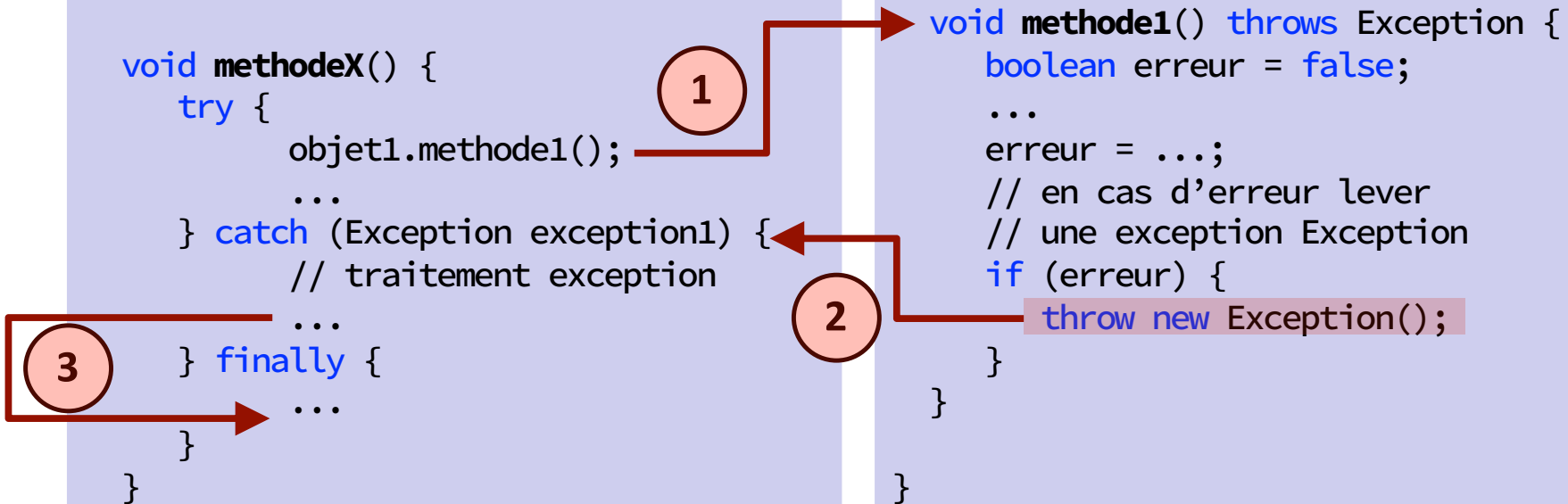


puis `finally`





# Propagation d'une exception

```
class Classe2 {  
    Classe1 objet1 = new Classe1();  
    ...  
    void methodeX() {  
        try {  
            objet1.methode1();  
            ...  
        } catch (Exception exception1) {  
            // traitement exception  
            ...  
        } finally {  
            ...  
        }  
    }  
    void methodeY() throws Exception {  
        objet1.methode1();  
        ...  
    }  
}
```

```
class Classe1 {  
    ...  
    void methode1() throws Exception {  
        boolean erreur = false;  
        ...  
        erreur = ...;  
        // en cas d'erreur lever  
        // une exception Exception  
        if (erreur) {  
            throw new Exception();  
        }  
    }  
}
```



## Exécution levée

-  `methodeX()` appelle `objet1.methode1()`
-  `methode1()` lève une `Exception`
-  `methodeX()` intercepte & traite `Exception`
-  puis `finally`

# Propagation d'une exception

```
class Classe2 {  
  
    Classe1 objet1 = new Classe1();  
    ...  
  
    void methodeX() {  
        try {  
            objet1.methode1();  
            ...  
        } catch (Exception exception1) {  
            // traitement exception  
            ...  
        } finally {  
            ...  
        }  
    }  
  
    void methodeY() throws Exception {  
        objet1.methode1();  
        ...  
    }  
}
```

```
class Classe1 {  
  
    ...  
  
    void methode1() throws Exception {  
        boolean erreur = false;  
        ...  
        erreur = ...;  
        // en cas d'erreur lever  
        // une exception Exception  
        if (erreur) {  
            throw new Exception();  
        }  
    }  
}
```



## Exécution normale



methodeY() appelle objet1.methode1()



methode1() se déroule sans problème



puis suite de methodeY()

# Propagation d'une exception

```
class Classe2 {  
  
    Classe1 objet1 = new Classe1();  
    ...  
  
    void methodeX() {  
        try {  
            objet1.methode1();  
            ...  
        } catch (Exception exception1) {  
            // traitement exception  
            ...  
        } finally {  
            ...  
        }  
    }  
  
    void methodeY() throws Exception {  
        objet1.methode1();  
        ...  
    }  
}
```

```
class Classe1 {  
    ...  
  
    void methode1() throws Exception {  
        boolean erreur = false;  
        ...  
        erreur = ...;  
        // en cas d'erreur lever  
        // une exception Exception  
        if (erreur) {  
            throw new Exception();  
        }  
    }  
}
```

vers l'appelant



## Exception levée



methodeY() appelle objet1.methode1()



methode1() lève une Exception



methodeX() propage Exception vers appelant