
Java, les objets : tout de suite !

Gestion des erreurs : les exceptions

Cnam Paris

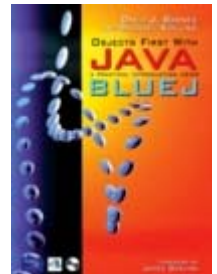
jean-michel Douin, douin@cnam.fr

Version de Juin 2003

Notes de cours associées au chapitre 11
tutorial BlueJ

<http://www.bluej.org/doc/documentation.html>

Ce support accompagne, référence le livre de David J. Barnes & Michael Kölling
Objects First with Java A Practical Introduction using BlueJ
Pearson Education, 2003 ISBN 0-13-044929-6.



Sommaire

- **Programmation défensive**

ou

- **comment développer un programme tolérant aux fautes**

toute “erreur” devrait être prise en compte,

Quelle décision prendre ?

Pas de changement d'état et retour à l'état antérieur

Signaler cette erreur à l'appelant

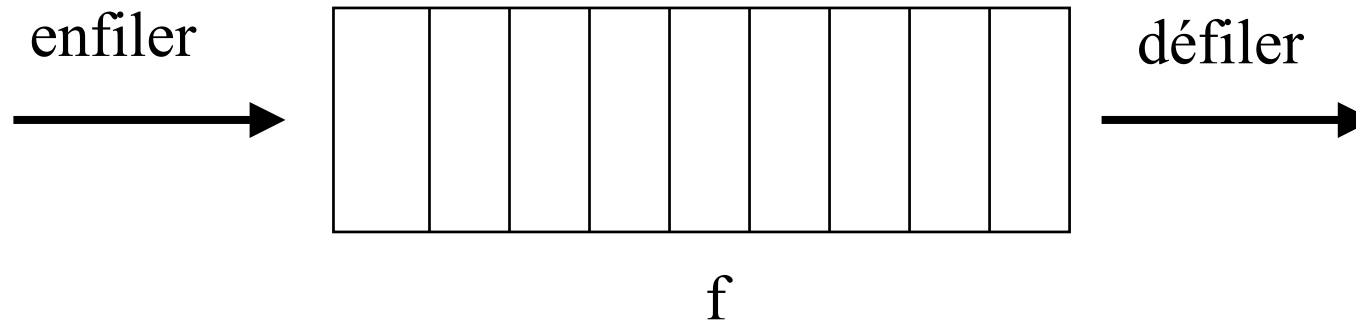
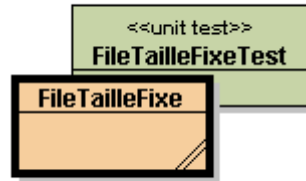
- **Vocabulaire**

Erreur ou Exception

Erreur signalée ou Exception levée ...

Exception filtrée ...

Un exemple : une file d' "Object" de taille fixe



- **Un classique : premier entré, premier sorti**

```
f.enfiler(new Integer(5));  
f.enfiler(new Integer(6));
```

```
Integer x = f.défiler(); assert x.intValue() == 6;  
Integer y = f.défiler(); assert y.intValue() == 5;
```

Les services proposés

Field Summary

static int	TAILLE_MINIMALE la taille minimale est de 2
------------	--

Constructor Summary

[FileTailleFixe\(\)](#)
création d'une file de la taille minimale, voir TAILLE_MINIMALE.

[FileTailleFixe\(int capacité\)](#)
Création d'une file de la taille souhaitée.

Method Summary

java.lang.Object	défiler() Défiler un élément.
void	enfiler(java.lang.Object elt) Enfiler un élément.
boolean	estPleine() test si la file est pleine.
boolean	estVide() test si la file est videe.
boolean	invariant() invariant de représentation.

Le constructeur ListeTailleFixe

Constructor Summary

[FileTailleFixe\(\)](#)

création d'une file de la taille minimale, voir TAILLE_MINIMALE.

[FileTailleFixe\(int capacité\)](#)

Création d'une file de la taille souhaitée.

Cas d'erreur possibles sur la capacité demandée par l'utilisateur

capacité < 0

capacité == 0 ? (est-ce une erreur ?)

capacité >= MAX, mais quelle valeur pour MAX ?

Les méthodes ajouter et lire

public void enfiler(Object elt)

cas d'erreurs

la file est pleine

public Object dépiler()

cas d'erreurs

la file est vide

Choix d'implémentation ajouter et lire

Hypothèse sur les éléments : ils sont différents de null
enfiler(null) n'est pas possible

public void enfiler(Object elt)

cas d'erreurs

si la file est pleine **aucun effet**

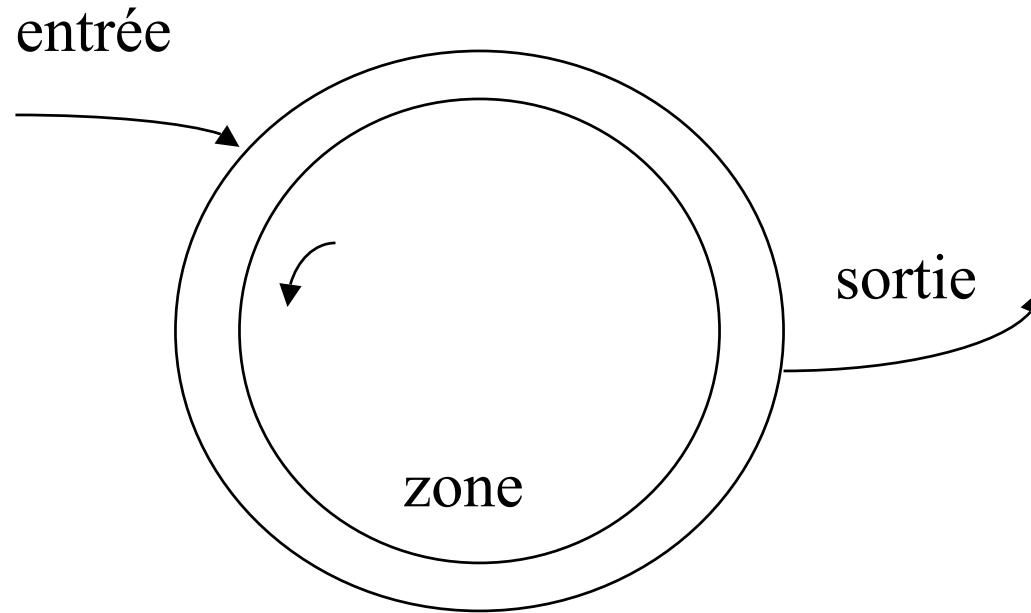
si elt est null **aucun effet**

public Object dépiler()

cas d'erreurs

si la file est vide la valeur **null est retournée**

Choix de représentation



- **File de taille fixe : un tableau géré de façon circulaire**

Une première implémentation de la file

```
public class FileTailleFixe{

    public static final int TAILLE_MINIMALE = 2;
    private Object[] zone;
    private int      entrée, sortie;
    private int      taille;

    public FileTailleFixe(){
        this(TAILLE_MINIMALE);
    }

    public FileTailleFixe(int capacité){
        zone = new Object[ Math.max(capacité, TAILLE_MINIMALE) ];
        entrée = sortie = taille = 0;
    }
}
```

Capacité de la file toujours > 0

! Pas assez de mémoire : le programme s'arrête

Une première implémentation de la file(fin)

```
public void enfiler(Object elt){
    if (!estPleine() && elt != null){
        zone[entrée] = elt;
        entrée = (entrée+1) % zone.length;
        taille++;
    }
}
```

```
public Object défiler(){
    if( !estVide()){
        Object elt = zone[sortie];
        sortie = (sortie +1) % zone.length;
        taille--;
        return elt;
    }else{
        return null;
    }
}
```

```
public boolean estPleine(){ return taille == zone.length; }
public boolean estVide(){ return taille == 0; }
```

exemple possible de code pour l'appelant

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

    int taille = Integer.parseInt(args[0]);
    FileTailleFixe f = new FileTailleFixe(taille);

    if(!f.estPleine()) f.enfiler(new Integer(3));
    if(!f.estPleine()) f.enfiler(new Integer(4));
    if(!f.estPleine()) f.enfiler(new Integer(5));

    Integer elt = (Integer)f.défiler();
    if(elt != null) System.out.println(" elt : " + elt);

    if(!f.estVide()){
        Integer elt2 = (Integer)f.défiler();
        if(elt2 != null) System.out.println(" elt2 : " + elt2);
    }
}
```

Toutes les erreurs sont-elles traitées ?

non : si args[0] ne contient pas un nombre et args.length == 0?

Constatations

- **Le code pour l'utilisateur est fastidieux**
- **les tests seront vite délaissés, oubliés...**

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

    int taille = Integer.parseInt(args[0]); // <-- erreurs possibles ici
    FileTailleFixe f = new FileTailleFixe(taille);

    f.enfiler(new Integer(3));
    f.enfiler(new Integer(4));
    f.enfiler(new Integer(5)); // <-- ici la file peut être pleine

    System.out.println(" elt : " + f.défiler().toString());
    System.out.println(" elt : " + f.défiler().toString());
    System.out.println(" elt : " + f.défiler().toString()); //<-- ici
                                                                // une erreur se produit
                                                                // si la file est vide
```

Erreur ou Exception

- **Nécessité de traiter les cas anormaux <<standard>>**

Division par zéro, débordement de tableaux, etc...

- **Nécessité d'un mécanisme de déclenchement d'une erreur (les exceptions)**

Se concentrer sur l'essentiel

Alléger le source du programme (afin d'éviter les tests fastidieux)

Transférer la responsabilité du traitement de l'erreur à l'appelant

Levées d'exceptions : 3 exemples

- **Division par zéro**

`class java.lang.ArithmeticException`

- **Débordement de tableau**

`class java.lang.ArrayIndexOutOfBoundsException`

- **Mauvais format de nombre**

`class java.lang.NumberFormatException`

- ...

- **Et aussi file vide, file pleine ...**

Exception : une division par zéro

```
2 |
3 | public class ExceptionStandard{
4 |     public static void main(String[] args){
5 |         int x = Integer.parseInt(args[0]);
6 |         int y = Integer.parseInt(args[1]);
7 |
8 |         int z = x / y;
9 |
10 |        System.out.println(" z = x / y " + z);
11 |    }
```

```
>java -cp . ExceptionStandard 6 2
```

```
Exception in thread "main" java.lang.ArithmeticException
    at ExceptionStandard.main(ExceptionStandard.java:8)
```

try/catch : présentation

- Condition **anormale** d'exécution d'un programme

```
try {
```

```
    instructions;
```

```
    instructions;
```

```
    instructions;
```

```
} catch( ExceptionType1 e){
```


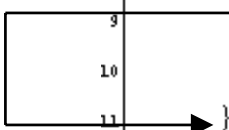
```
    traitement de ce cas anormal (de type ExceptionType1);
```

```
}
```

L'exception est filtrée

```
3 public class ExceptionStandard{
4     public static void main(String[] args){
5         int x = Integer.parseInt(args[0]);
6         int y = Integer.parseInt(args[1]);
7
8         try{
9             int z = x / y;
10            System.out.println(" z = x / y " + z);
11        } catch(ArithmeticException ae){
12            System.out.println(" erreur : division par 0 !");
13        }
14
15    }
16 }
17 }
18 }
```

si y==0



BlueJ: Terminal Window

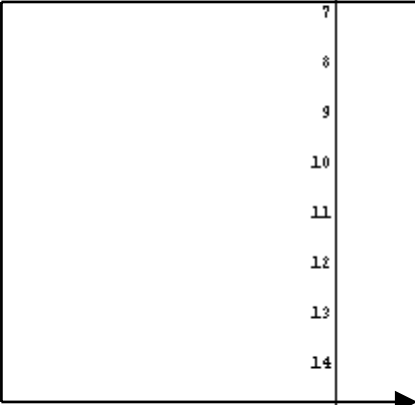
Options

erreur : division par 0 !

Exception : un débordement de tableau

```
2 |
3 | public class ExceptionStandard{
4 |     public static void main(String[] args){
5 |         try{
6 |             int x = Integer.parseInt(args[0]);
7 |             int y = Integer.parseInt(args[1]);
8 |
9 |             try{
10 |                 int z = x / y;
11 |                 System.out.println(" z = x / y " + z);
12 |             }catch(ArithmeticException ae){
13 |                 System.out.println(" erreur : division par 0 !");
14 |             }
15 |         }catch(ArrayIndexOutOfBoundsException e){
16 |             System.out.println(" deux arguments sont attendus");
17 |         }
18 |
```

si args.length == 1



The diagram consists of a rectangular box that encloses lines 6 through 14 of the code. An arrow originates from the right side of this box and points to the opening curly brace of the second catch block on line 15. This visualizes that the condition 'si args.length == 1' is satisfied, leading to the execution of the code within the box, which eventually results in an ArrayIndexOutOfBoundsException being thrown.

try/catch : présentation

try {

instructions;

instructions;

instructions;

} catch(*ExceptionType1* e){

traitement de ce cas anormal de type ExceptionType1;

} catch(*ExceptionType2* e){

traitement de ce cas anormal de type ExceptionType2;

}

Exception : un mauvais format de nombre

```
3 public class ExceptionStandard{
4     public static void main(String[] args){
5         try{
6             int x = Integer.parseInt(args[0]);
7             int y = Integer.parseInt(args[1]);
8
9             try{
10                int z = x / y;
11                System.out.println(" z = x / y " + z);
12            }catch(ArithmeticException ae){
13                System.out.println(" erreur : division par 0 !");
14            }
15        }catch(ArrayIndexOutOfBoundsException e){
16            System.out.println(" deux arguments sont attendus");
17        }catch(NumberFormatException nfe){
18            System.out.println("mauvais format du nombre " + nfe.getMessage());
19        }
20    }
```

BlueJ: Terminal Window

Options

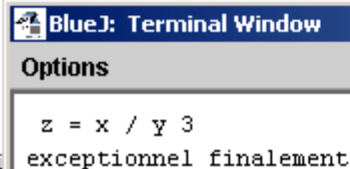
mauvais format du nombre 0A

try/catch/finally : présentation

```
try {  
    instructions;  
    instructions;  
    instructions;  
} catch( ExceptionType1 e){  
    traitement de ce cas anormal de type ExceptionType1;  
} catch( ExceptionType2 e){  
    traitement de ce cas anormal de type ExceptionType2;  
  
} finally (  
    traitement de fin de bloc try ;  
}
```

finally{ } : dès l'entrée dans le bloc try/catch

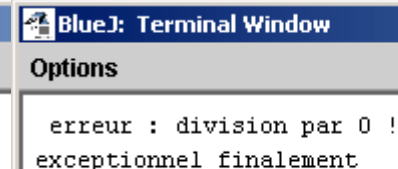
```
5  try{
6      int x = Integer.parseInt(args[0]);
7      int y = Integer.parseInt(args[1]);
8
9      try{
10         int z = x / y;
11         System.out.println(" z = x / y " + z);
12     }catch(ArithmeticException ae){
13         System.out.println(" erreur : division par 0 !");
14     }
15 }catch(ArrayIndexOutOfBoundsException e){
16     System.out.println(" deux arguments sont attendus");
17 }catch(NumberFormatException nfe){
18     System.out.println("mauvais format du nombre " + nfe.getMessage());
19 }finally{
20     System.out.println("exceptionnel finalement");
21 }
```



BlueJ: Terminal Window

Options

```
z = x / y 3
exceptionnel finalement
```



BlueJ: Terminal Window

Options

```
erreur : division par 0 !
exceptionnel finalement
```

try/catch/throw : présentation

try {

instructions;

instructions;

instructions;

} catch(*ExceptionType1* e){

traitement de ce cas anormal de type ExceptionType1;

} catch(*ExceptionType2* e){

traitement de ce cas anormal de type ExceptionType2;

throw e; // l'exception est propagée

// vers le bloc try/catch) englobant

} finally (

traitement de fin de bloc try ;

}

Propagation d'une exception

```
5     try{
6         int x = Integer.parseInt(args[0]);
7         int y = Integer.parseInt(args[1]);
8
9         try{
10            int z = x / y;
11            System.out.println(" z = x / y " + z);
12        }catch(ArithmeticException ae){
13            System.out.println(" erreur : division par 0 !");
14        }
15    }catch(ArrayIndexOutOfBoundsException e){
16        System.out.println(" deux arguments sont attendus");
17    }catch(NumberFormatException nfe){
18        System.out.println("mauvais format du nombre " + nfe.getMessage());
19        throw nfe;
20    }finally{
21        System.out.println("exceptionnel finalement");
22    }
23
```

```
>java -cp . ExceptionStandard 0 0A
```

```
mauvais format du nombre 0A
```

```
exceptionnel finalement
```

```
Exception in thread "main" java.lang.NumberFormatException: 0A
```

```
    at java.lang.Integer.parseInt(Integer.java:435)
```

```
    at java.lang.Integer.parseInt(Integer.java:476)
```

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

Le try/catch de la machine installé par défaut

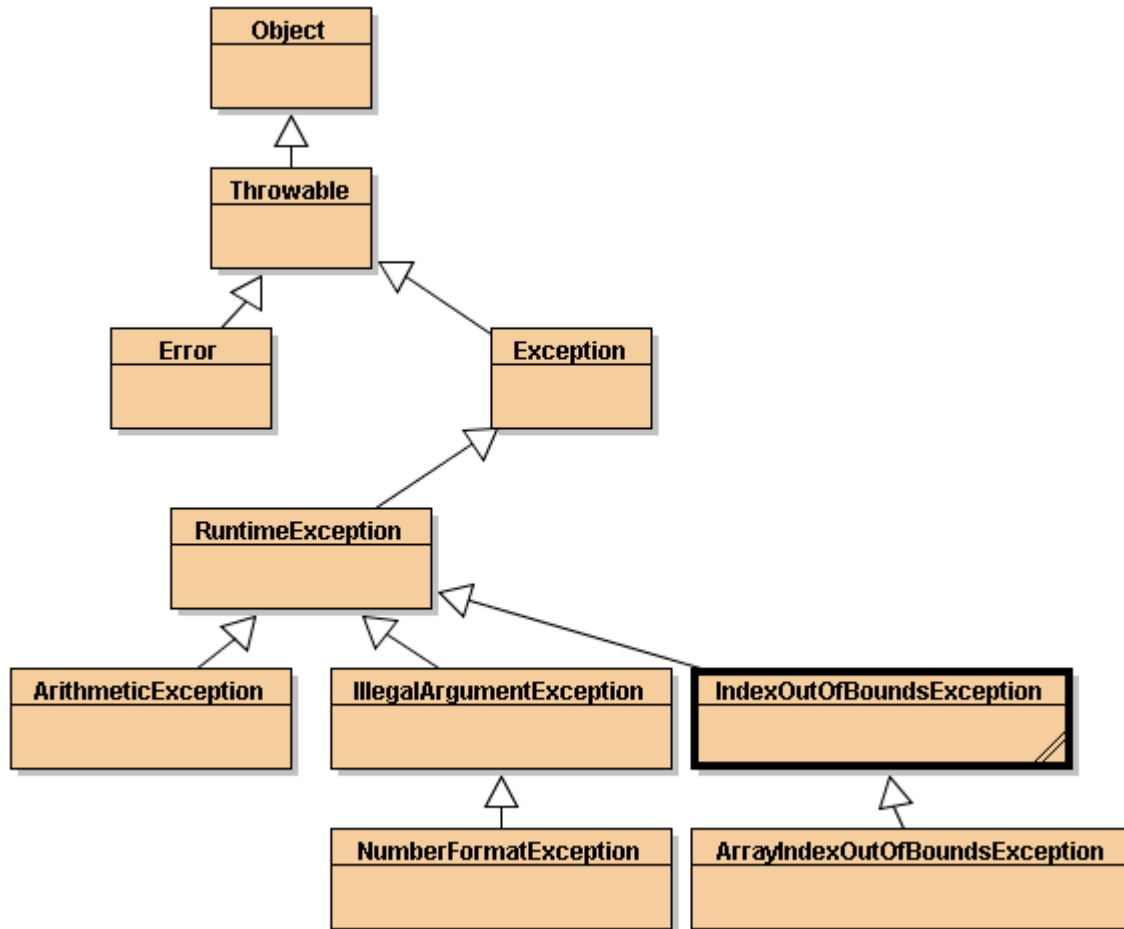
- **try{**

```
try{
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);

    try{
        int z = x / y;
        System.out.println(" z = x / y " + z);
    }catch(ArithmeticException ae){
        System.out.println(" erreur : division par 0 !");
    }
}catch(ArrayIndexOutOfBoundsException e){
    System.out.println(" deux arguments sont attendus")
}catch(NumberFormatException nfe){
    System.out.println("mauvais format du nombre " + nfe);
    throw nfe;
}finally{
    System.out.println("exceptionnel finalement");
}
```

- **}catch(Throwable t){**
- **System.err.println(t.printStackTrace());**
- **}**

Grappe d'héritage (un extrait)

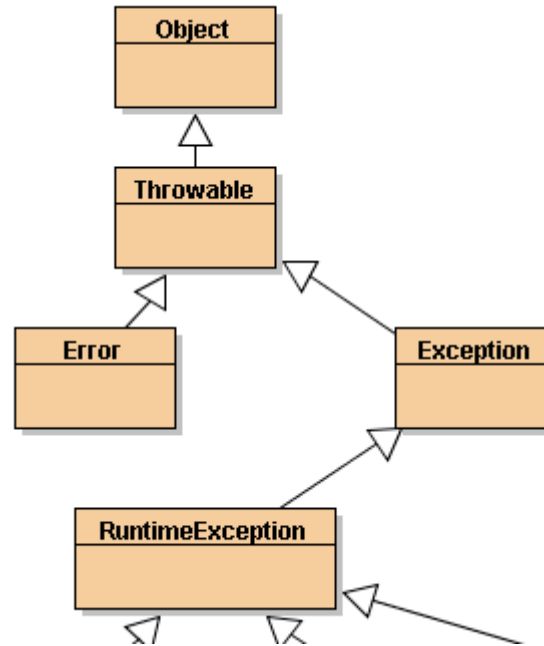


La classe Throwable

Method Summary

Throwable	fillInStackTrace() Fills in the execution stack trace.
Throwable	getCause() Returns the cause of this throwable or null if the cause is nonexistent or unknown.
String	getLocalizedMessage() Creates a localized description of this throwable.
String	getMessage() Returns the detail message string of this throwable.
StackTraceElement []	getStackTrace() Provides programmatic access to the stack trace information printed by printStackTrace() .
Throwable	initCause(Throwable cause) Initializes the <i>cause</i> of this throwable to the specified value.
void	printStackTrace() Prints this throwable and its backtrace to the standard error stream.
void	printStackTrace(PrintStream s) Prints this throwable and its backtrace to the specified print stream.
void	printStackTrace(PrintWriter s) Prints this throwable and its backtrace to the specified print writer.
void	setStackTrace(StackTraceElement[] stackTrace) Sets the stack trace elements that will be returned by getStackTrace() and printed by printStackTrace() and related methods.
String	toString() Returns a short description of this throwable.

Error, Exception héritent de Throwable



- **RuntimeException**

Les exceptions « RuntimeException »

- **try/catch n'est pas obligatoire**
(Heureusement !) pour toutes les classes dérivées

ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IndexOutOfBoundsException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, SecurityException, SystemException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationException

- **De toute façon**
Toutes les exceptions sont prises en compte par le try/catch englobant de la machine

Créer une exception

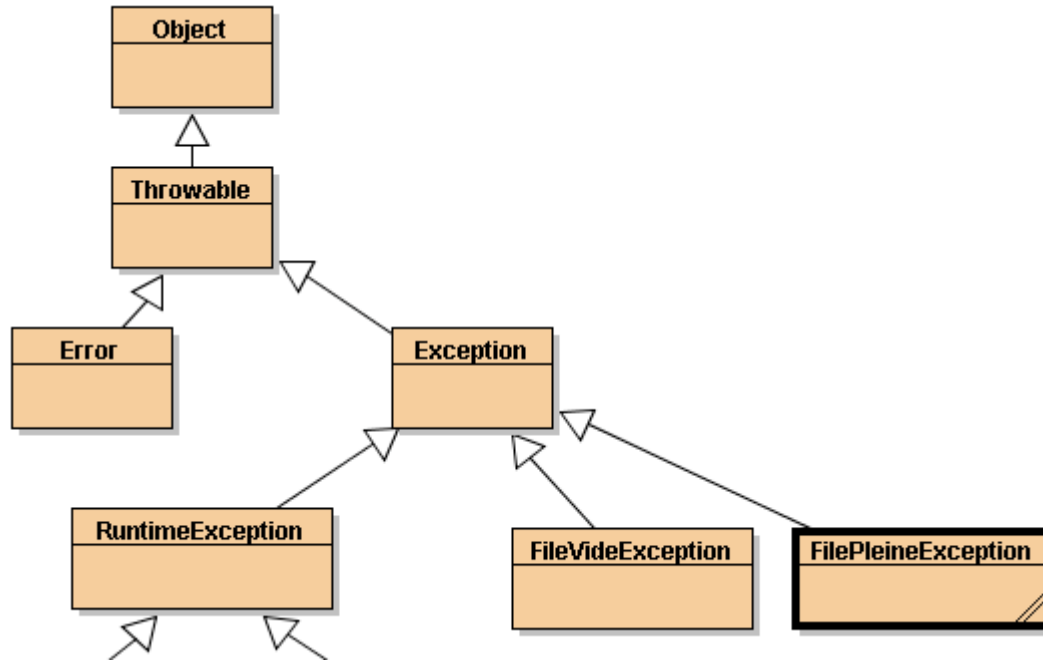
- **Traitement des cas anormaux**
- **La file est pleine et l'utilisateur tente d'enfiler**
- **La file est vide et l'utilisateur tente de défiler**
- **Ces exceptions sont des classes dérivées de la classe Exception**
 - Des contraintes légitimes seront imposées à l'utilisateur**

Nouvelles Exceptions

```
public class FilePleineException extends Exception{  
  
}
```

```
public class FileVideException extends Exception{  
  
}
```

Le graphe



Levée d'une exception, throw et throws

Méthode enfiler(Object elt)

```
    if (estPleine()) throw new FilePleineException(); // throw

    zone[entrée] = elt;
    entrée = (entrée+1) % zone.length;
    taille++;
}
}
```

public void enfiler(Object elt) throws FilePleineException{ // throws

```
    if (estPleine()) throw new FilePleineException();

    zone[entrée] = elt;
    entrée = (entrée+1) % zone.length;
    taille++;
}
}
```

La clause throws

- **Précise les exceptions susceptibles d'être levées (ou propagées) par cette méthode**
- **L'appelant doit préciser ce qu'il veut faire de cette exception**

try/catch

et/ou

**ajout à la clause throws,
dans ce cas c'est à l'appelant de l'appelant de
préciser ce qu'il ...**

Pour l'utilisateur

```
74 public static void main(String[] args){
75     int taille = Integer.parseInt(args[0]);
76     FileTailleFixeAvecExceptions f = new FileTailleFixeAvecExceptions(taille);
77     try{
78         f.enfiler(new Integer(3));
79         f.enfiler(new Integer(4));
80         f.enfiler(new Integer(5));
81
82         Integer elt = (Integer)f.défiler();
83         System.out.println(" elt : " + elt);
84         Integer elt2 = (Integer)f.défiler();
85         System.out.println(" elt2 : " + elt2);
86     }catch(FilePleineException fpe){
87         System.out.println("la file est pleine !");
88     }catch(FileVideException fpe){
89         System.out.println("la file est vide !");
90     }
91 }
```

- Il en manque : `args[0]`, `args.length`

Pour l'utilisateur (2)

```
75     try{
76         int taille = Integer.parseInt(args[0]);
77         FileTailleFixeAvecExceptions f = new FileTailleFixeAvecExceptions(taille);
78
79         f.enfiler(new Integer(3));
80         f.enfiler(new Integer(4));
81         f.enfiler(new Integer(5));
82
83         Integer elt = (Integer)f.défiler();
84         System.out.println(" elt : " + elt);
85         Integer elt2 = (Integer)f.défiler();
86         System.out.println(" elt2 : " + elt2);
87     }catch(FilePleineException fpe){
88         System.out.println("la file est pleine !");
89     }catch(FileVideException fpe){
90         System.out.println("la file est vide !");
91     }catch(RuntimeException re){
92         re.printStackTrace();
93     }
```

- Toutes les instances des classes dérivées de RuntimeException

Pour l'utilisateur(3)

```
75     try{
76         int taille = Integer.parseInt(args[0]);
77         FileTailleFixeAvecExceptions f = new FileTailleFixeAvecExceptions(taille);
78
79         f.enfiler(new Integer(3));
80         f.enfiler(new Integer(4));
81         f.enfiler(new Integer(5));
82
83         Integer elt = (Integer)f.défiler();
84         System.out.println(" elt : " + elt);
85         Integer elt2 = (Integer)f.défiler();
86         System.out.println(" elt2 : " + elt2);
87     } catch(Exception e){
88         e.printStackTrace();
89     }
```

- 
- Toutes les instances des classes dérivées de Exception

Pour l'utilisateur JASS

- **Une autre démarche**

- 1) **Établir les pré, post assertions, invariant de classe**
- 2) **développer le code**

L'analyse par JASS apporte

Deux versions de sources,

le source initial (aux commentaires pertinents)

et un source instrumenté dans un style plutôt défensif

Pour l'utilisateur ESC/Java

- Allez en

http://jfod.cnam.fr/tp_cdi/douin/ESC_JAVA.pdf

La file est au transparent 97, annexe1

Une discussion s'impose ...

NB : Les assertions du jdk1.4

```
3 public class AssertionsCatch{
4
5     public AssertionsCatch(int taille){
6         assert taille > 0;
7     }
8
9     public static void main(String[] args){
10        try{
11            new AssertionsCatch(3);
12            new AssertionsCatch(0);
13        }catch(AssertionError ae){
14            ae.printStackTrace();
15        }
16    }
```

```
>java -cp . -ea AssertionsCatch
```

```
java.lang.AssertionError
```

```
    at AssertionsCatch.<init>(AssertionsCatch.java:6)
```

```
    at AssertionsCatch.main(AssertionsCatch.java:12)
```

Mais, encore une source d'erreurs !

- **f.enfiler(new Integer(4));**
- **f.enfiler(new Boolean(true));**

- **Integer i = (Integer)f.défiler();**

- **Une solution consiste en la manipulation de file homogènes**
 - Généricité ou polymorphisme paramétrique**
 - Vérification statique effectuée par le compilateur**
- **Généricité présente dans le jdk1.5**
 - Voir jsr14 adding generics**

Résumé, synthèse
