



Département économique IESN

2^{ème} baccalauréat en
Informatique de Gestion

Syllabus de Java

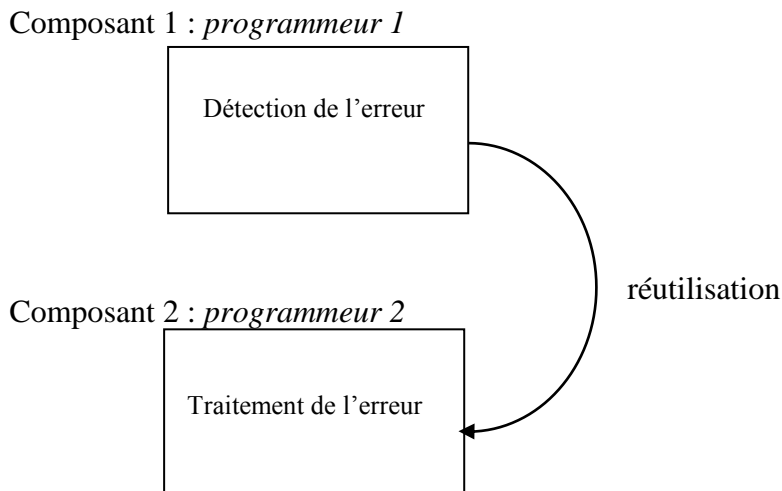
Françoise Dubisy

Année académique 2015-2016

Chapitre 1 : La gestion des cas d'erreur : les exceptions

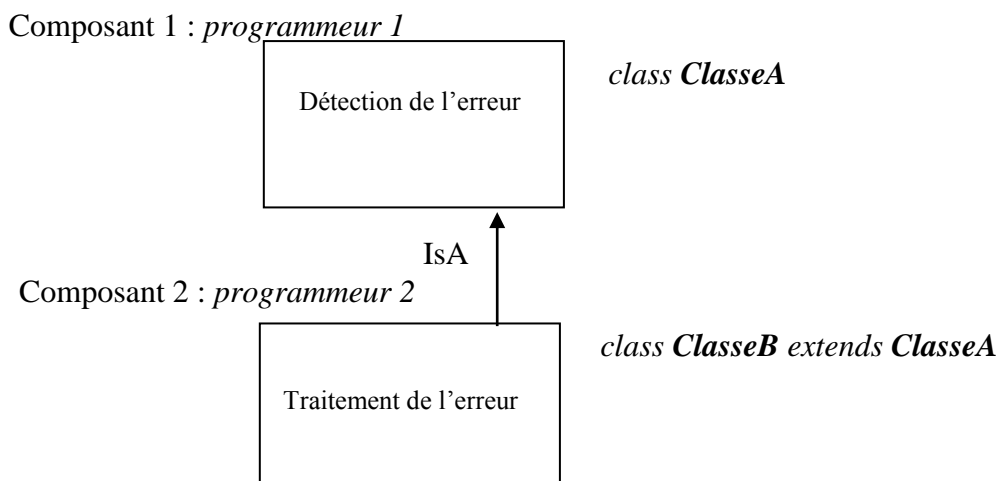
La gestion des cas d'erreurs en utilisant le mécanisme des exceptions semble a priori lourd. Il faut en comprendre tout l'intérêt. Pour rappel, un des points forts de la programmation orientée objet est la possibilité de réutiliser et d'adapter des composants existants. Un composant 1 écrit par un programmeur 1 peut donc être réutilisé (intégré) dans un composant 2 écrit par un autre programmeur (programmeur 2) .

Le principe des exceptions est de permettre la **séparation de la détection** d'un incident (un problème ou un cas d'erreur) **de sa prise en charge**. Le cas d'erreur peut être détecté dans le composant 1 et être traité dans le composant 2 qui fait appel au composant 1. En effet, c'est le programmeur 2 qui peut décider comment doit réagir le programme lorsqu'un cas d'erreur se présente : ex : arrêter le programme, lancer une procédure particulière, envoyer un message d'erreur à l'utilisateur via une boîte de dialogue, afficher un simple message sur la console, ...



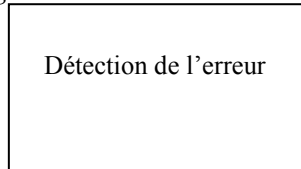
Exemples de réutilisation:

A. Le composant 2 est une **sous-classe** du composant 1 :



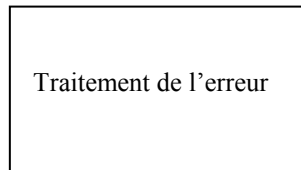
- B. Le composant 1 est une classe (*ClasseA*) importée dans le composant 2 qui est la classe *Principal*. Une occurrence (*occ*) de la classe *ClasseA* est créée dans la classe *Principale* et des méthodes de la classe *ClasseA* sont appelées sur cet objet *occ*.

Composant 1 : *programmeur 1*



```
class ClasseA
{ ...
...methodeX(...) {...}
...
}
```

Composant 2 : *programmeur 2*



```
class Principal
{... main...
{ClasseA occ = new ClasseA(...);
...occ.methodeX(...)
...
}
}
```

Grâce au mécanisme des exceptions, le concepteur du composant 1 peut écrire le code de **détection du cas d'erreur sans prendre de décision quant au traitement de l'erreur**. Le composant 1 se contente alors **d'avertir, le cas échéant**, le composant appelant (composant 2) qu'une erreur a été détectée. Le programmeur du composant appelant (composant 2) peut alors **recupérer l'erreur détectée** par le composant 1 et **décider de son traitement** (arrêter le programme, lancer une procédure particulière, envoyer un message d'erreur à l'utilisateur ...).

1.1. Détecter l'erreur: lancer une exception

Lorsqu'un cas d'erreur est détecté dans le composant 1, **un objet de type Exception est généré** par le composant 1. Cet objet peut mémoriser des informations sur l'erreur qui a eu lieu. Une classe de type *Exception* doit donc être créée par le programmeur 1 (cfr section 7.2).

Soit la classe *Personne* (composant 1) écrite par le programmeur 1. Un objet de type *personne* est caractérisé entre autres par une variable d'instance de type caractère représentant son sexe (valeur 'm' ou 'f'). Un cas d'erreur pourrait être détecté dans la classe *Personne* si une tentative est effectuée pour affecter une valeur autre que 'f' ou 'm' à la variable d'instance *sexe*.

Pour éviter toute incohérence dans la valeur affectée à cette variable d'instance, le concepteur de la classe *Personne* (programmeur 1) déclare privée (**private**) la variable d'instance *sexe*. La seule possibilité pour affecter une valeur à la variable *sexe* si l'on se trouve en dehors de la classe *Personne* est donc d'utiliser le setter **public** correspondant (**setSexe(...)**).

Le test de **détection d'une valeur erronée** que l'on tenterait d'attribuer à la variable *sexe* est à placer **dans ce setter**. Si la valeur que l'on tente d'attribuer à la variable d'instance *sexe* est erronée, un objet de type *SexeException* est généré. La classe *SexeException* doit être écrite (cfr section 7.2).

```
package pack1;
public class Personne

{ private String nomPre;
  private char sexe;

  public void setSexe (char s)      ...
  { if (s != 'f' && s != 'm')
    throw new SexeException(s);
    else sexe = s;
  }
}
```

Le code à écrire pour générer un objet de type *SexeException* est :

throw new SexeException(...);

NB. *to throw* en anglais signifie lancer.

Le constructeur de la classe *SexeException* reçoit en argument la valeur erronée que l'on a tenté d'attribuer à la variable *sexe*. Cet argument permet de mémoriser dans l'exception la valeur erronée, ce qui pourrait être utile par exemple, pour construire un message d'erreur personnalisé (càd reprenant la valeur erronée).

Le mécanisme d'exception que l'on met ainsi en place est destiné à avertir l'utilisateur du composant 1 qu'un cas d'erreur a été détecté. Or, le programmeur du composant 2 qui utilisera le composant 1 n'a ***pas forcément accès au code*** du composant qu'il utilise. Il peut par contre consulter la ***documentation*** accompagnant la classe qu'il utilise. Cette documentation contient en général la *déclaration des variables* (d'instance et de classe) ainsi que la déclaration des *constructeurs* et *méthodes* déclarés public.

Il faut donc qu'à la seule lecture de ***la déclaration d'une méthode*** qu'il compte utiliser, le programmeur sache si la méthode est *susceptible* ou non de *détecter une erreur*, autrement dit si la méthode est susceptible de *générer une exception*, et si oui, *de quel type*.

Ceci explique que la déclaration d'une méthode susceptible de détecter un cas d'erreur doit contenir le mot réservé **throws** suivi du nom de la classe correspondant au type d'exception éventuellement générée.

Attention : ne pas confondre les mots réservés *throw_* et *throws_*. Le mot *throw* s'utilise dans le code de la méthode, le mot *throws* dans sa déclaration!

La déclaration complète de la méthode *setSexe* est donc :

public void **setSexe** (char s) **throws SexeException**

Le code de la classe *Personne* est donc:

```
public class Personne
{ private String nomPre;
  private char sexe;
  public void setSexe (char s) throws SexeException
  { if (s != 'f' && s != 'm')
    throw new SexeException(s);
    else sexe = s;
  } }
```

Attention : **dès qu'un cas d'erreur est détecté** et qu'un objet de type exception est créé, **le reste de la méthode est abandonné**. Autrement dit, dès qu'une *instruction contenant un throw est exécutée*, on sort de la méthode et les instructions qui suivraient éventuellement l'instruction *throw* ne sont pas exécutées.

1.2. Création d'une classe de type Exception

Toute classe utilisée pour générer des exceptions doit être une **sous-classe de la classe *Exception* ou d'une de ses sous-classes**.

Toute classe d'exception peut contenir ses propres variables d'instance. Une variable d'instance qu'il serait intéressant de prévoir est une variable permettant de stocker la valeur erronée qu'on a tenté d'affecter. Cette valeur pourrait être obtenue via le constructeur. La mémorisation de la valeur erronée dans l'objet de type exception a pour avantage par exemple de construire un message d'erreur personnalisé, c'est-à-dire rappelant la valeur erronée. Ce message d'erreur peut être construit dans la méthode `toString()` ou dans la méthode héritée `getMessage()`.

```
package pack1;
```

```
public class SexeException extends Exception
```

```
{ private char sexeErrone;
```

```
    public SexeException (char sE)  
    { sexeErrone = sE; }
```

```
    public String toString() // ou redéfinition de la méthode héritée getMessage()  
    { return "La valeur " + sexeErrone + " proposée pour le sexe est invalide!"; }  
}
```

1.3. Propager l'erreur : relancer l'exception

Le programmeur qui fait appel à une méthode susceptible de détecter une erreur c-à-d de générer une exception (méthode qui comprend une clause *throws* dans sa déclaration) doit choisir parmi les deux solutions suivantes :

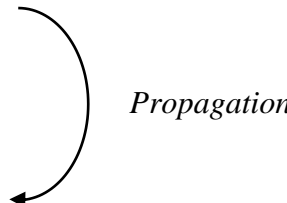
- soit **traiter** le cas d'erreur (traiter l'exception) ;
- soit **propager** l'exception vers le niveau appelant supérieur.

La section 7.4. aborde le traitement des erreurs. Nous nous concentrons dans la section 7.3. sur la propagation des exceptions.

Si le programmeur qui fait appel à une méthode susceptible de détecter une erreur ne désire **pas traiter l'erreur**, il peut décider de **reporter cette décision vers le niveau appelant**. On dit alors qu'il y a **propagation** de l'exception. Le mécanisme de la propagation d'exception se déroule comme suit. Une **methodeA** qui fait appel dans son code à une **methodeB** contenant une clause *throws* dans sa déclaration doit recopier cette clause *throws* dans sa propre déclaration.

```
... methodeB(...) throws ExceptionX
{ if (...)
    throw new ExceptionX(...);
  ...
}

... methodeA (...) throws ExceptionX
{ ...
  ...methodeB(...);
  ...
}
```



C'est le cas par exemple lorsque le concepteur de la classe *Personne* (programmeur 1) a prévu que le constructeur de la classe *Personne* fasse appel à la méthode *setSexe(...)* alors que celle-ci est susceptible de lancer une exception de type *SexeException*. Le concepteur de la classe *Personne* peut décider de laisser à l'appréciation du futur utilisateur de cette classe (programmeur 2) la décision du traitement de l'erreur.

Le programmeur 1 se contente alors de **recopier dans la déclaration du constructeur** la clause *throws* de la méthode *setSexe(...)*, à savoir, ***throws SexeException***.

Le code de la classe *Personne* ainsi complété devient :

```
public class Personne

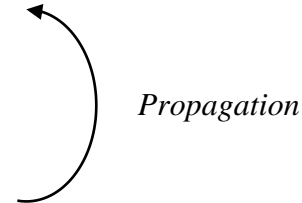
{private String nomPre;
 private char sexe;

 public Personne(String n, char s) throws SexeException
 { nomPre = n;
   setSexe(s);
 }

 public void setSexe (char s) throws SexeException
 { if (s != 'f' && s != 'm')
     throw new SexeException(s);
   else sexe = s;
 }

 public String toString()
 {return "La personne " + nomPre;}

}
```



1.4. Traiter l'erreur : capturer l'exception

Si le programmeur qui fait appel à une méthode susceptible de détecter une erreur désire **traiter l'erreur**, il doit placer *l'appel à cette méthode* dans un **bloc try** suivi d'un **bloc catch** précisant la **réaction à avoir en cas d'erreur** (to try signifie *essayer* et to catch signifie *attraper* en anglais). Le mot réservé *catch* est suivi du nom de la classe correspondant au *type d'exception que l'on essaye d'attraper*. On place alors dans le bloc **catch** le *code* spécifiant la façon de *réagir en cas d'erreur* correspondant à ce type d'exception.

Le bloc *try* peut contenir plusieurs instructions. Dès qu'une instruction génère une exception, le code du bloc *catch* correspondant est exécuté **mais le reste des instructions du bloc try ne sont pas exécutées.**

Prenons comme exemple la classe *Principal* (composant 2 écrit par le programmeur 2). La classe *Personne* du package *pack1* est importée dans la classe *Principal*.

Le code de la méthode *main* de la classe *Principal* tente de créer une occurrence de la classe *Personne*. Pour ce faire, il faut faire appel au constructeur de la classe *Personne* dont la déclaration est:

```
public Personne(String n, char s) throws SexeException
```

Ce constructeur est susceptible de générer une exception de type *SexeException*. Son appel est donc à placer dans un bloc **try**.

Ce bloc est suivi d'un bloc **catch** qui tente d'attraper une occurrence de type *SexeException*. Si une occurrence de type *SexeException* existe, c'est qu'elle a été générée lors de l'exécution de la méthode *setSexe(...)* de la classe *Personne*, et donc qu'un cas d'erreur a été détecté. Si une occurrence de type *SexeException* a été générée par la méthode *setSexe(...)*, la référence de cette occurrence est placée automatiquement par java dans la variable **e** (cfr *catch (SexeException e)*).

Le traitement de l'erreur consiste ici en l'ouverture d'une boîte de dialogue qui affiche le message correspondant à l'erreur. Ce message est obtenu en appelant implicitement la méthode *toString()* sur l'objet **e** (de type *SexeException*).

```
package pack2 ;
import pack1.*;
import javax.swing.*; // si utilisation de composant swing
                    // ex: JOptionPane (boîte de dialogue)

public class Principal

{ public static void main(String[ ] args)
  { try
    { Personne p = new Personne ("J. Petit",'m');      (1)
      System.out.println(p);                          (2)
    }
    catch (SexeException e)
    { JOptionPane.showMessageDialog(null, e, "Erreur", JOptionPane.ERROR_MESSAGE);
    }
  }
  (3)
  System.exit(0) ;
}}
```

Appel implicite au *toString()* sur
l'objet **e** de type *SexeException*

Si l'appel au constructeur (cfr (1)) génère une exception, le **reste du bloc try est abandonné** : l'instruction (2) n'est pas exécutée. Ce qui convient parfaitement, étant donné que si l'appel au constructeur a généré une exception, l'objet *p* n'a pas été créé et donc pas initialisé, et par conséquent, l'exécution de l'instruction (2) provoquerait une erreur : il est en effet impossible d'appeler implicitement la méthode *toString()* sur un objet qui n'existe pas (**p** est une **référence nulle**).

Ceci explique que placer l'instruction (2) en dehors du bloc *try*, après le bloc *catch* (en (3)), constitue une erreur de programmation détectée à la compilation. Le compilateur affiche alors le message d'erreur stipulant que l'objet *p* pourrait être **non initialisé**.

En conclusion, **tout objet que l'on tente de créer (donc d'initialiser) dans un bloc try ne peut être utilisé en dehors de ce bloc.**

1.5. Classes d'exception existantes

Il existe de nombreuses classes d'exception disponibles. Ces classes sont organisées en une hiérarchie d'héritage. La racine de cette hiérarchie est la classe *Exception*.

Bon nombre de méthodes dans les classes créées par les concepteurs de Java ont dans leur déclaration une clause *throws* signalant que ces méthodes sont susceptibles de générer des exceptions. L'appel à ces méthodes doit donc être placé dans un bloc *try*.

Le bloc *catch* associé peut faire appel aux méthodes **disponibles dans la classe d'exception** correspondante (cfr documentation de la classe d'exception apparaissant dans la déclaration de la méthode appelée).

Parmi ces méthodes, deux paraissent intéressantes :

- la méthode **getMessage()** à appeler sur un objet de type *Exception* et qui retourne une **chaîne de caractères proposant une description de l'erreur** détectée (à utiliser via l'instruction *System.out.println* ou comme message à afficher dans une boîte de dialogue par exemple);
- la méthode **printStackTrace()** à appeler sur un objet de type *Exception* (ne retourne rien) qui affiche directement dans la fenêtre console le contenu de la pile des erreurs rencontrées (plus difficile à déchiffrer).

Exemple :

```
package pack3;

public class ClassX
{
    ...
    public ... methodeX ( ... ) throws ExceptionX
    {
        if ( ... )
            throw new ExceptionX(...);
        else
            ...
    }
    ...
}
```



```
package pack4;

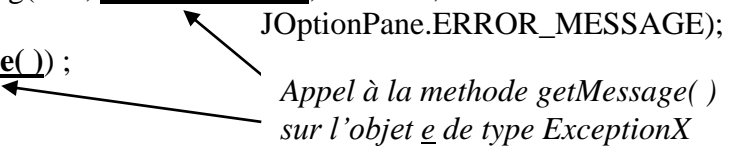
import pack3.*;
import javax.swing.*;

public class Principal

{ public static void main(String[ ] args)

    { try
      { ClassX x = new ClassX(...);
        ... x.methodeX(...);
        ...
      }
      catch (ExceptionX e)
      { JOptionPane.showMessageDialog(null, e.getMessage(), "Erreur",
                                         JOptionPane.ERROR_MESSAGE);
//ou : System.out.println(e.getMessage()) ;
//ou : e.printStackTrace() ;

      }
    }
}
```



1.6 Exceptions et héritage

Si une sous-classe fait appel à une méthode de la super-classe qui est susceptible de générer une exception, il faut ***soit propager l'exception*** (en recopiant la clause *throws* dans la déclaration de la méthode appelante) ***soit la traiter*** (en plaçant l'appel à la méthode dans un bloc *try*).

Il en va de même pour les constructeurs. Toute sous-classe fait appel dans son constructeur au constructeur de la super-classe via l'instruction *super(...)*. Si le constructeur de la super-classe est susceptible de générer une exception, il contient donc une clause *throws* dans sa déclaration. Le constructeur de la sous-classe doit soit propager l'exception, soit la traiter.

Soit la classe *Etudiant* qui est une sous-classe de la classe *Personne*.

Le constructeur de la classe *Etudiant* fait appel au constructeur de la classe *Personne* via l'instruction *super(...)*.

Or, le constructeur de la classe *Personne* est susceptible de générer une exception. La déclaration du constructeur de *Personne* est en effet :

```
public Personne(String n, char s) throws SexeException
```

Le constructeur de la classe *Etudiant* doit donc *soit traiter l'erreur, soit la propager*. Dans l'exemple ci-dessous, le constructeur de la classe *Etudiant* propage l'exception. Sa déclaration contient donc la clause : **throws SexeException**

```
public class Etudiant extends Personne
```

```
{ private String section;  
  private int annee;
```

```
  public Etudiant(String n, char sex, String sect, int a) throws SexeException
```

```
  { super(n, sex);  
    section = sect;  
    annee = a;  
  }  
}
```

← Appel au constructeur de la classe *Personne*
qui est susceptible de générer une exception

1.7 Plusieurs exceptions lancées/générées par la même méthode

Une même méthode peut détecter plus d'un type d'erreur. Si c'est le cas, sa déclaration doit contenir la clause **throws** suivi du nom de chacune des classes d'exception correspondant aux exceptions susceptibles d'être générées par la méthode :

... **methodeM** (...) **throws** **ExceptionX**, **ExceptionY**, **ExceptionZ**, ...

Rappelons que *dès qu'une erreur est détectée lors de l'exécution d'une méthode et qu'un objet de type exception est par conséquent généré, le reste de la méthode n'est pas exécuté*. Ceci implique qu'à l'exécution, **une méthode ne peut détecter qu'un seul cas d'erreur**. En effet, dès qu'un objet de type exception est généré, l'exécution du reste de la méthode est stoppée : une même exécution de méthode ne peut donc pas générer plus d'un objet de type exception à la fois.

A titre d'exemple, on pourrait prévoir la gestion de cas d'erreur sur l'année d'inscription d'un étudiant : les valeurs permises seraient 1, 2 ou 3. Toute tentative d'affecter une valeur autre que celles-ci serait considérée comme cas d'erreur. Le même principe que pour la gestion des valeurs affectées à la variable *sexe* est appliqué ici. La variable *annee* est déclarée **private** et un **setter public** est prévu : *setAnnee*(...). Ce setter étant susceptible de générer une exception de type **AnneeException**, sa déclaration contient la clause **throws** **AnneeException**. La classe *AnneeException*, sous-classe de la classe *Exception*, doit être écrite. Le constructeur de la classe *Etudiant* doit donc faire appel au setter *setAnnee*(...) pour affecter la valeur reçue en argument à la variable *annee*. La déclaration du constructeur doit donc signaler que des exceptions de deux classes différentes sont susceptibles d'être générées. Sa déclaration contient donc la clause : **throws** **SexeException**, **AnneeException**

```
public class Etudiant extends Personne
```

```
{ private String section;  
  private int annee;
```

```
  public Etudiant(String n, char sex, String sect, int a)  
    throws SexeException, AnneeException  
  { super(n, sex);  
    section = sect;  
    setAnnee(a);  
  }
```

```
public void setAnnee (int a) throws AnneeException
{ if ( a < 1 || a > 3) throw new AnneeException (a);
  else
    annee = a;
}

public String toString( )
    {return super.toString( ) + " est inscrite en " + annee +"e" + section;}
}
```

```
public class AnneeException extends Exception

{ private int anneeErronee;

  public AnneeException(int a)
    {anneeErronee = a;}

  public String toString( )
    {return "La valeur " + anneeErronee + " proposée pour l'année est invalide!";}
}
```

1.8. Traiter plusieurs erreurs : capturer plusieurs exceptions

Lorsqu'un programmeur fait appel à une méthode dont la déclaration contient une clause *throws* portant sur plusieurs classes de type exception, il doit comme convenu placer cet appel dans un bloc *try*. Ce bloc *try* peut être suivi de plusieurs blocs *catch* : un bloc *catch* par type d'exception à traiter. Chaque bloc *catch* contient alors le code correspondant à la façon de traiter un type d'erreur.

```
try {...
    ... methodeM ( ... ) ;
    ...
}
catch (ExceptionX e)
    {... /* reaction en cas d'erreur de type ExceptionX */
}
catch (ExceptionY e)
    {... /* reaction en cas d'erreur de type ExceptionY */
}

catch (ExceptionZ e)
    {... /* reaction en cas d'erreur de type ExceptionZ */
}
```

Soit la classe *Etudiant* telle que proposée à la section 7.7.

La classe *Principal* essaye de créer une occurrence de la classe *Etudiant*. La déclaration du constructeur de la classe *Etudiant* est :

```
public Etudiant(String n, char sex, String sect, int a)
    throws SexeException, AnneeException
```

Le constructeur de la classe *Etudiant* est donc susceptible de générer des exceptions de deux types : *SexeException* et *AnneeException*. Si l'on désire réagir différemment à ces deux cas d'erreur, on prévoit deux blocs *catch* dans la classe *Principal*. Le premier bloc *catch* précise comment réagir si la valeur proposée pour le **sexe** est invalide (on affiche ici le message correspondant à l'erreur via l'instruction *System.out.println*). Le second bloc *catch* précise comment réagir si la valeur proposée pour l'**année** est invalide (on affiche ici le message correspondant à l'erreur via une boîte de dialogue).

```
import javax.swing.* ;
public class Principal

{ public static void main(String[ ] args)

    { try
      { Etudiant etu = new Etudiant ("J. Petit",'m', "info",3);
        System.out.println(etu);
      }
      catch (SexeException e)
      { System.out.println(e) ;
      }
      catch (AnneeException e)
      { JOptionPane.showMessageDialog(null, e, "Erreur: année",
                                     JOptionPane.ERROR_MESSAGE);
      }
      System.exit(0);
    }
}
```

NB. Ne pas oublier l'instruction **System.exit(o)**; en fin de programme lorsqu'on utilise des boîtes de dialogue.

Notons que si on désire réagir de la même façon lorsque l'on tente d'afficher une valeur erronée à la variable *sexe* ou à la variable *annee*, il n'est pas nécessaire de prévoir deux blocs *catch*. On pourrait par exemple afficher le message d'erreur via une boîte de dialogue, qu'il s'agisse d'une erreur sur la variable *sexe* ou d'une erreur sur la variable *annee*. Un seul bloc *catch* suffit alors, à condition que la classe d'exception capturée soit une super-classe des classes d'exception *SexeException* et *AnneeException*. Un seul bloc *catch* est prévu qui tente de capturer une exception de type *Exception* (càd une occurrence de la classe *Exception*) :

```
catch (Exception e)
```

Le titre de la boîte de dialogue est *erreur* et le contenu est fourni par l'appel implicite à la méthode *toString()* sur l'objet *e*. Le mécanisme du **polymorphisme** s'appliquera sur l'objet *e* : le contenu du message d'erreur sera personnalisé en fonction du type d'erreur.

```
import javax.swing.* ;

public class Principal

{ public static void main(String[ ] args)

    { try
      { Etudiant etu = new Etudiant ("J. Petit",'m', "info",3);
        System.out.println(etu);
      }
      catch ( Exception e)
      { JOptionPane.showMessageDialog(null, e, "Erreur",
                                     JOptionPane.ERROR_MESSAGE);
      }
      System.exit(0);
    }
}
```

1.9. La clause finally

Les blocs *try* et *catch* peuvent être complétés par un bloc *finally*. Celui-ci est placé après le dernier bloc *catch*.

Les instructions contenues dans le bloc *finally* seront exécutées qu'il y ait eu ou non une exception générée. Les instructions du bloc *finally* seront donc exécutées :

- à la suite des instructions du bloc *try*, s'il n'y a pas d'erreur,
- à la suite des instructions d'un bloc *catch*, en cas d'erreur.

```
try { ...
    ... methodeM ( ... ) ;
    ...
}
catch (ExceptionX e)
{ ... /* réaction en cas d'erreur de type ExceptionX */
}
catch (ExceptionY e)
{ ... /* réaction en cas d'erreur de type ExceptionY */
}

catch (ExceptionZ e)
{ ... /* réaction en cas d'erreur de type ExceptionZ */
}

finally
{ ... /* instructions à exécuter dans tous les cas */
}
```

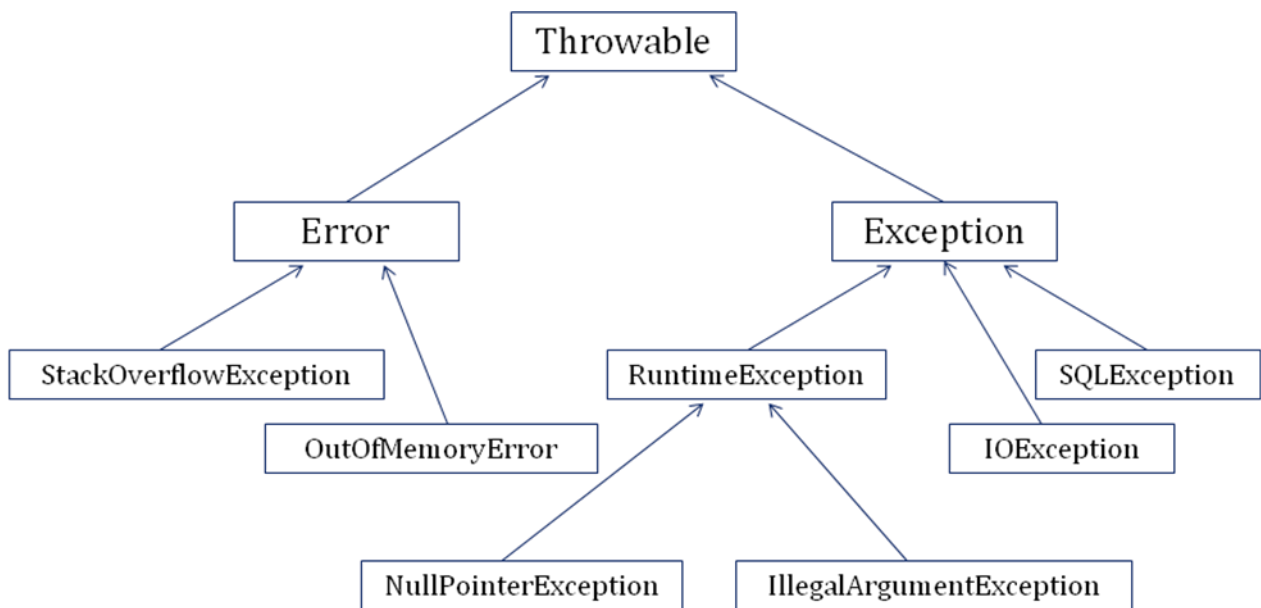
1.10. Les exceptions non contrôlées

Les exceptions non contrôlées sont des sous-classes de la classe **Error** ou de la classe **RuntimeException**.

Elles peuvent ne pas être gérées, c'est-à-dire pas de try-catch ni de propagation.

De plus, elles ne doivent pas forcément être déclarées dans la clause throws des méthodes qui sont susceptibles de les lancer.

Le schéma suivant propose un extrait de la hiérarchie des exceptions :



Quand utiliser les exceptions non contrôlées ?

Ceci est laissé à l'appréciation du programmeur.

En vue de prendre la décision, on peut se poser la question suivante :

Quand l'erreur survient, est-ce que le code appelant peut réagir pour arranger la situation?

Si la réponse est oui, on utilise des exceptions contrôlées.

Si la réponse est non, cela signifie qu'il peut s'agir d'une erreur de programmation (ex : *NullPointerException*) et que le code appelant ne peut rien faire pour y remédier. On utilise alors des exceptions non contrôlées.

1.11. En résumé

① Le principe des exceptions est de permettre la **séparation de la détection** d'un incident (un problème ou un cas d'erreur) **de sa prise en charge**. La détection du cas d'erreur peut avoir lieu dans un composant 1 mais le traitement de ce cas d'erreur peut être réalisé dans un composant appelant le composant1.

② Pour gérer un cas d'erreur via le mécanisme des exceptions, il faut créer une **sous-classe d'une classe d'Exception**.

Ex : `class ClasseException extends Exception`

Lorsqu'un cas d'erreur est détecté, il faut **créer une occurrence** de cette sous-classe d'exception via l'instruction `throw`.

Ex : `throw new ClasseException(...)`

③ La **méthode** qui est **susceptible de détecter l'erreur** et donc de générer un objet de type exception doit contenir **dans sa déclaration la clause `throws`** suivi du nom de la classe d'exception correspondante.

Ex : `... methodeX (...) throws ClasseException`

④ **Dès qu'un cas d'erreur est détecté** lors de l'exécution d'une méthode (et, par conséquent, qu'un objet de type exception est créé), l'exécution de la méthode est stoppée : **le reste de la méthode est abandonné**.

⑤ Toute méthode appelant une méthode susceptible de générer une exception doit :

- soit propager l'exception,
- soit traiter l'exception.

⑥ Propager l'exception consiste à **reporter la décision du traitement de l'erreur vers le niveau appelant**. Le mécanisme de la propagation d'exception se fait comme suit. Une methodeY qui fait appel dans son code à une methodeX contenant une clause throws dans sa déclaration doit **recopier cette même clause `throws` dans sa propre déclaration**:

Ex :

```
... methodeX(...) throws ClasseException
    { if (...) throw new ClasseException (...);
    ... }
... methodeY(...) throws ClasseException
    { ...methodeX(...);
    ... }
```

Propagation

⑦ *Traiter l'exception* signifie décider de la réaction à avoir en cas d'erreur (ex : arrêter le programme, lancer une procédure particulière, envoyer un message d'erreur à l'utilisateur, ...). Pour traiter l'exception, il faut placer *l'appel à la méthode* susceptible de générer un objet de type exception **dans un bloc try** et placer **dans un bloc catch** le code spécifiant la **façon de réagir à l'erreur**.

```
Ex :
try { ...
    ... methodeY ( ... ) ;
    ...
}
catch (ClasseException e)
{ ...          /* réaction en cas d'erreur */
}
```

⑧ Le bloc *try* peut contenir plusieurs instructions. Dès qu'une instruction génère une exception, le code du bloc *catch* correspondant est exécuté **mais le reste des instructions du bloc try ne sont pas exécutées**.

⑨ Si une **sous-classe** fait appel à une méthode de la **super-classe** qui est susceptible de générer une exception, il faut **soit propager l'exception** (en recopiant la clause *throws* dans la déclaration de la méthode appelante) **soit la traiter** (en plaçant l'appel à la méthode dans un bloc *try*).

⑩ Une même méthode peut détecter plus d'un type d'erreur. Si c'est le cas, sa déclaration doit contenir la clause **throws suivi du nom de chacune des classes d'exception** correspondant aux exceptions susceptibles d'être générées par la méthode :

```
Ex : ... methodeY ( ... ) throws ClasseExceptionA, ClasseExceptionB, ...
```

⑪ L'appel à une méthode dont la déclaration contient une clause **throws portant sur plusieurs classes de type exception** doit être placé dans un bloc *try*. Ce bloc *try* peut être suivi de **plusieurs blocs catch** : un bloc *catch* par type d'exception à traiter. Chaque bloc *catch* contient alors le code correspondant à la façon de traiter un type d'erreur.

```
Ex :
try { ...
    ... methodeY ( ... ) ;
    ...
}
catch (ClasseExceptionA e)
{ ...          /* reaction en cas d'erreur de type ClasseExceptionA */
}
catch (ClasseExceptionB e)
{ ...          /* reaction en cas d'erreur de type ClasseExceptionB */
}
```

⑫ Un bloc **finally** peut être placé après le dernier bloc *catch*. Les instructions du bloc *finally* sont exécutées qu'il y ait ou non une exception générée.

①③ Les exceptions non contrôlées (sous-classes de la classe `Error` ou `RuntimeException`) peuvent ne pas être gérées. Elles ne doivent pas obligatoirement être déclarées dans la clause `throws` de la signature des méthodes.

Chapitre 1 : La gestion des cas d'erreur : les exceptions

1.1. Détecter l'erreur : lancer une exception	3
1.2. Création d'une classe de type Exception	5
1.3. Propager l'erreur : relancer l'exception	5
1.4. Traiter l'erreur : capturer l'exception	7
1.5. Classes d'exception existantes	8
1.6. Exceptions et héritage	9
1.7. Plusieurs exceptions lancées/générées par la même méthode	10
1.8. Traiter plusieurs erreurs : capturer plusieurs exceptions	11
1.9. La clause <i>finally</i>	13
1.10. Les exceptions non contrôlées	14
1.11. En résumé	15