Année 2008-2009

esil

Les exceptions Java

Nicolas Baudru
mél : nicolas.baudru@esil.univmed.fr
page web : nicolas.baudru.perso.esil.univmed.fr

Supposez que vous écrivez un programme qui accède à un fichier, ou qui se connecte à un serveur.

Que se passerait-il si le fichier disparaît, ou si le serveur tombe en panne ? Le comportement de votre programme serait au mieux interrompu, au pire deviendrait imprévisible.

L'idéal serait de pouvoir écrire du code qui gèrerait ces situations exceptionnelles. C'est ce que nous allons voir dans ce cours.

JavaSound est une collection de classes et d'interfaces Java faisant partie de la bibliothèque de classe standard de J2SE. Elle est composée de deux parties :

- MIDI (Musical Instrument Digital Interface).
- et Sampled.

Un fichier MIDI peut être vu comme une partition de musique : les données MIDI ne sont pas des sons mais des instructions indiquant comment un instrument MIDI peut restituer des sons.

Ici nous allons nous servir d'un instrument logiciel fourni dans Java : le synthétiseur, correspondant à un objet Java de type Sequencer.

Un programme qui ne compile pas

esil

```
import javax.sound.midi.*;

public class MiniMusiqueApp {
  public static void main (String [] args) {
    MiniMusiqueApp mini = new MiniMusiqueApp();
    mini.jouer();
  }

  public void jouer() {
    Sequencer lecteur = MidiSystem.getSequencer();
    System.out.println("Nous␣avons␣un␣sequenceur");
  }
}
```

```
% javac MiniMusiqueApp.java
MiniMusiqueApp.java : 10 : unreported
                          exception javax.sound.midi.
MidiUnvailableException; must be caught
                          or declared to be thrown
...
```

Explication

[1] Supposons que vous voulez appeler une méthode d'une classe de la bibliothèque standard Java.

[2] Cette méthode fait quelque chose de risqué, qui peut ne pas fonctionner au moment de l'exécution (ex : ouvrir un fichier, ouvrir une connexion réseau, endormir un processus, ...)

[3] Vous devez dire au compilateur que vous savez que la méthode que vous appelez est risquée.

[4] Vous écrivez alors du code qui gère l'erreur si elle se produit.

La gestion des exceptions Java est un mécanisme qui permet de traiter toutes les situations inattendues qui peuvent se produire au moment de l'exécution.

Elle s'appuie sur le fait que vous savez que la méthode peut avoir un comportement risqué. Si c'est le cas, vous devez écrire explicitement un morceau de code qui prend en compte cette éventualité.

Lorsqu'une méthode possède un comportement risqué, on dit qu'elle peut générer (ou lancer) une exception.

Comment savoir si une méthode lance une exception ?

Pour savoir si une méthode peut lancer une exception, vous devez consulter la javadoc, et vérifier si la méthode contient une clause throws dans sa déclaration.

Comment utiliser une méthode risqué

Nous avons déjà vu que le compilateur Java veut savoir que vous savez que vous utilisez une méthode risquée. Pour ce faire vous devez envelopper le code risquée dans un bloc try/catch.

```
try{
  // comportements risqués ...
}
catch {Exception ex} {
  // code pour essayer de récupérer l'erreur
}
```

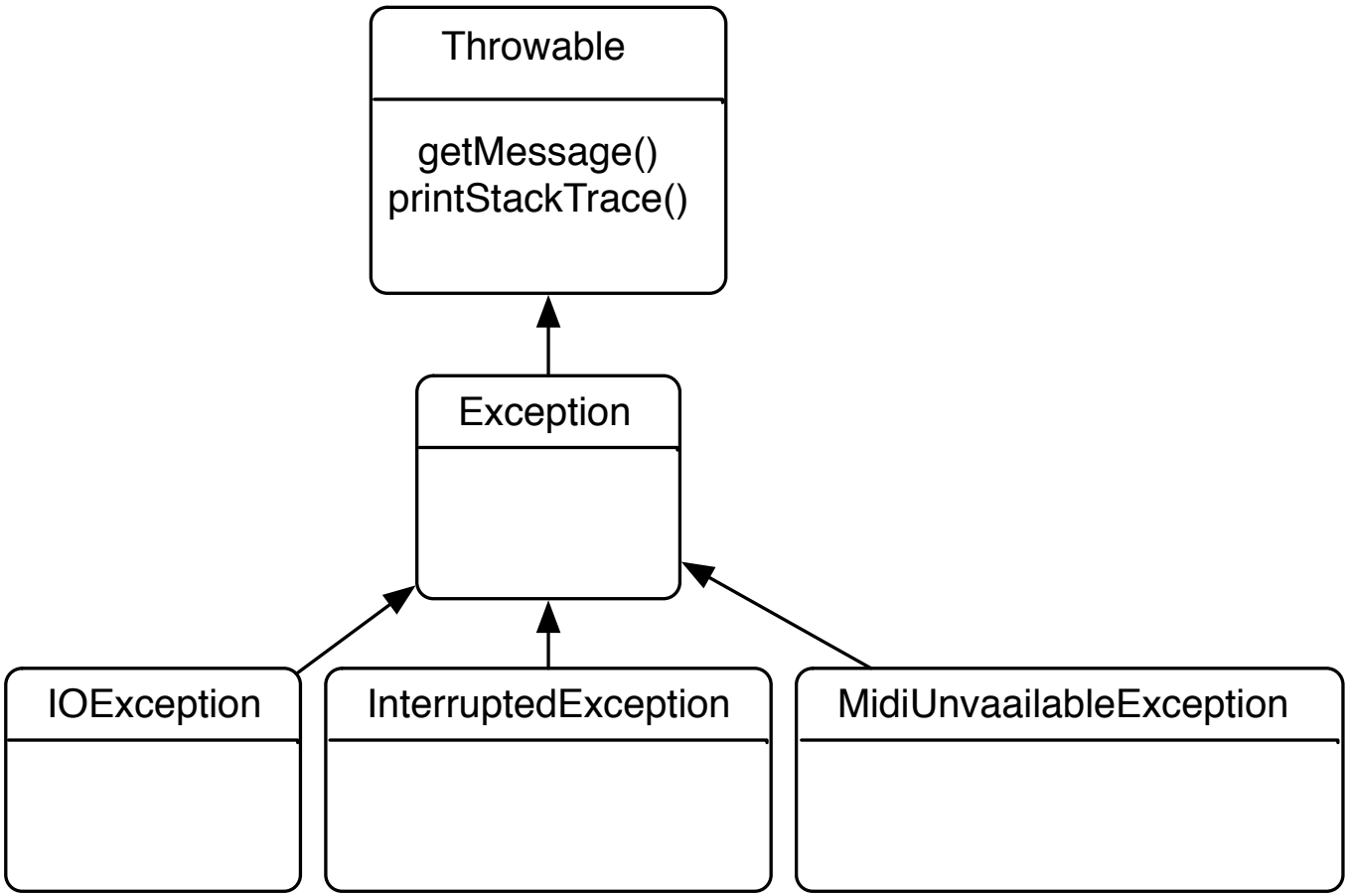
Ce qu'il faut mettre dans le bloc catch dépend de l'erreur à traiter.

Que feriez-vous si une exception était lancée lors d'une connexion à un serveur ?

Et lors de l'ouverture d'un fichier ?

Qu'est-ce que la variable ex ?

Une exception est un objet de type Exception. Et grâce au polymorphisme un objet de type Exception peut être une instance de nimporte quelle sous-classe d'Exception.



```
import javax.sound.midi.*;

public class MiniMusiqueApp {
  public static void main (String [] args) {
    MiniMusiqueApp mini = new MiniMusiqueApp();
    mini.jouer();
  }

  public void jouer() {
    try {
      Sequencer lecteur = MidiSystem.getSequencer();
      System.out.println("Nous avons un sequenceur");
    }
    catch (MidiAvailableException ex) {
      System.out.println("Aïe! un problème.");
    }
  }
}
```

Lancer et intercepter une exception

En général, un programmeur Java passera plus de temps à gérer les exceptions qu'à les créer. Bien que la plupart des méthodes lançant des exceptions font partie de la bibliothèque Java, il peut vous arriver de devoir écrire vous-même une méthode risquée.

Pour cela, il faut déclarer la méthode risquée comme étant Throwable (mot clé throws), puis dans le code de cette méthode, créer une nouvelle Exception puis la lancer (mot clé throw).

```
// quelque part ...
public void prendreRisque () throws MauvaiseException {
  if (SansEspoir) { throw new MauvaiseException (); }
}
// ailleurs dans le code
public void croiserLesDoigts () {
  try { unObjet.prendreRisque(); }
  catch (MauvaiseException ex) {
    ex.printStackTrace();
  }
}
```

Exceptions vérifiées ou non vérifiées par le compilateur

esil

Pour que votre programme compile :

- si vous lancez une exception dans votre code, vous devez la déclarer au moyen du mot-clé "throws" dans votre déclaration de méthode.
- Si vous appelez une méthode qui lance une exception, vous devez faire savoir au compilateur que vous savez que cette exception existe. Deux façons de faire :
  - utiliser un bloc try/catch pour gérer l'exception ;
  - choisir d'esquiver l'exception et de ne pas la gérer (voir plus loin).

Il existe cependant une exception à cette règle. Les RuntimeExceptions, (une sous-classe de la classe Exception) ne sont pas vérifiées par le compilateur. Il n'est donc pas nécessaire de les traiter.

Exemple : NullPointerException et DivideByException héritent de RuntimeExceptions et ne sont pas traitées par le compilateur..

Pourquoi le compilateur ne s'occupe-t-il pas de ce type d'exceptions ?

Un premier résumé

- Une méthode peut lancer une exception si quelque chose échoue au moment de l'exécution.
- Une exception est toujours un objet de type Exception.
- Les exceptions vérifiées par le compilateur doivent être traitées. Seules les RuntimeExceptions peuvent ne pas être traitées.
- Traiter une exception signifie ou bien la gérer (bloc try/catch) ou bien "l'éviter" (voir plus loin).
- Les méthodes qui peuvent lancer des exceptions doivent être déclarées "throws". Et l'exception peut alors être lancée à l'aide du mot clé "throw" suivi d'un nouvel objet de type exception.

```
throw new MaNouvelleException( );
```

Contrôle de flot dans un bloc try/catch

```
try {
   // 1)
   o.prendreUnRisque();
}
catch (Exception ex) {
   // 2)
   System.out.println("perdu␣!")
}
// 3)
 System.out.println("traitement␣␣terminé␣!");
 // ... le reste de la méthode
```

Règles :

- si le bloc 1) réussi, passer au bloc 3)
- si le bloc 1) échoue quelque part, passer au bloc 2), puis au bloc 3)

Contrôle de flot dans un bloc try/catch en utilisant finally

Si une portion de code doit être exécutée, qu'une exception se produise ou pas, vous pouvez la placer dans un bloc finally :

```
try {  // 1)
  o.prendreUnRisque();
}
catch (Exception ex) { // 2)
  System.out.println("perdu␣!")
}
finally { // 3)
  System.out.println("traitement␣␣terminé␣!")
}
// 4) ... le reste de la méthode
```

Règles :

- si le bloc 1) réussi, passer au bloc 3), puis au bloc 4)
- si le bloc 1) échoue, passer au bloc 2), puis au bloc 3), puis au bloc 4)
- si le bloc 1) ou 2) a une instruction return, 3) s'exécute quand même juste avant d'effectuer le return.

Une méthode peut lancer plusieurs exceptions

esil

```
public class Lessive {
  public void faireLaLessive() throws
  PantalonException, LingerieException {
    //  le code pouvant lancer des
    // PantalonExceptions ou des LingerieExceptions.
  }
}

// ... ailleurs
Lessive l = new Lessive();
try {  l.faireLaLessive () ;  }
catch ( PantalonException pex ) { ... }
catch ( LingerieException lex ) { ... }
```

Vous pouvez déclarer des exceptions en utilisant un supertype des exceptions que vous lancez. Il est alors possible de lancer des sous-classes d'exceptions sans les déclarer explicitement.

```
public void faireLaLessive ( ) throws VetementException {
  ...
  throw new PantalonException ();
  ...
  throw new LingerieException ();
}
```

Vous pouvez de la même manière intercepter des exceptions en utilisant un supertype de l'exception lancée.

```
try { l.faireLaLessive(); }
catch (VetementException vex) { // traitement}
```

Mais alors, pourquoi ne pas toujours récupérer un objet de type Exception ?

Ordonner les blocs catch

Si plusieurs types d'Exception sont possibles, et que vous souhaitez les traiter de manière différente, vous devez les récupérer en allant de la plus spécifique (dans l'héritage) à la plus générique.

```
Lessive l = new Lessive();
try {  l.faireLaLessive () ;  }
catch (PantalonVeloursException pvex) { ... }
catch ( PantalonException pex ) { ... }
catch ( LingerieException lex ) { ... }
catch (VetementException vex) { ... }
```

Esquiver une exception

Si vous ne voulez pas gérer une exception, vous pouvez l'esquiver en la déclarant.

Esquiver une exception signifie que votre méthode ne la traitera pas (pas de bloc try/catch pour cette exception). Votre méthode peut alors lancer indirectement une exception.

Par conséquent vous devez déclarer votre méthode throws, et ce sera à la méthode appelant votre méthode de gérer à son tour l'exception : le problème est reporté à plus tard.

```
public void uneMethode () throws MonException {
  l.faireLaLessive () ;
}
```

Quel est le type de l'exception lancée par votre méthode ?

Que se passe-t-il si tout le monde (même le main()) esquive l'exception ?

Try/catch et la pile

Il n'existe que deux façons de satisfaire le compilateur lors de l'utilisation d'une méthode risquée (qui lance une exception) :

1 gérer l'exception i.e. utiliser un bloc try/catch/finally.

2 l'esquiver et traiter le problème plus tard : vous devez alors déclarer votre méthode "throws", et laisser gérer l'exception à la méthode appelante.