Les exceptions

Rémi Forax
forax@univ-mlv.fr

Les Exceptions

- Mécanisme qui permet de reporter des erreurs vers les méthodes appelantes.
- Problème en C :
  - prévoir une plage de valeurs dans la valeur de retour pour signaler les erreurs.
  - Propager les erreurs “manuellement”
- En Java comme en C++, le mécanisme de remonté d’erreur est gérée par le langage.

Exemple d'exception

- Un exemple simple

```
public static void main(String[] args) {
  System.out.println(args[0]);
}
```

- Le bytecode (.class) contient
  - Le nom du fichier source
  - Le nom de chaque méthode et une table qui associe des offsets dans le code et des numéros de ligne

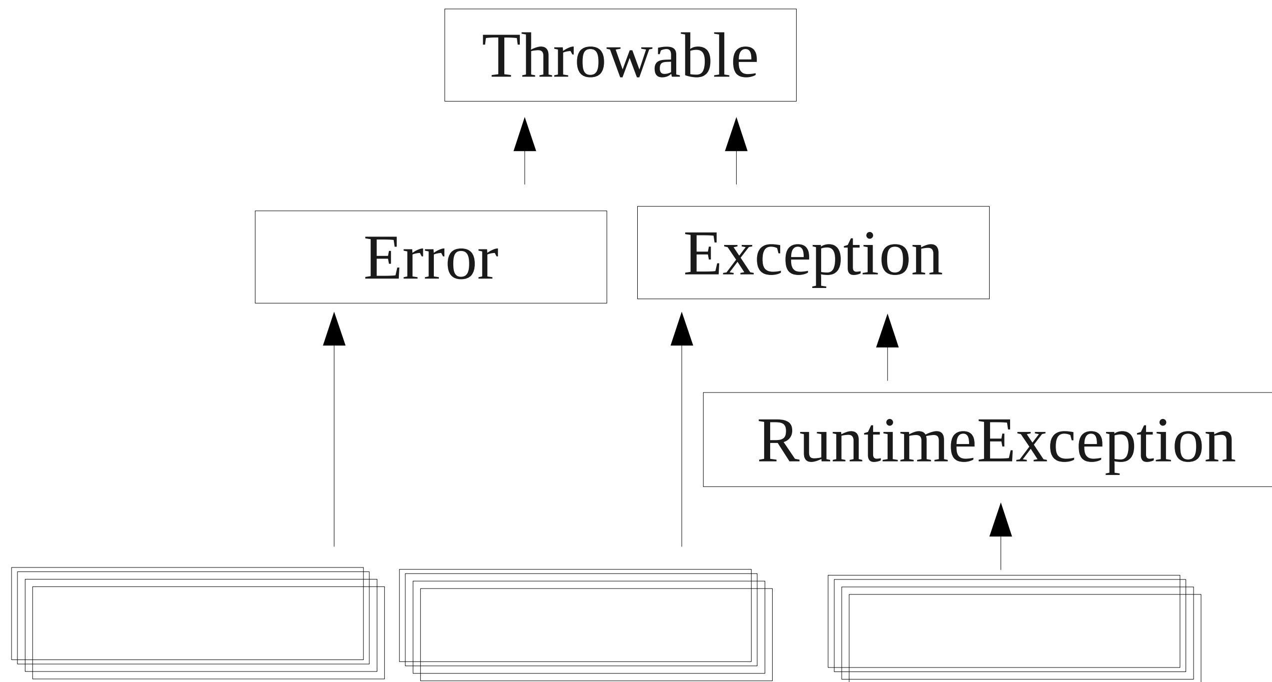
```
$ java ExceptionExample
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at ExceptionExample.main(ExceptionExample.java:8)
```

Comment ca marche ?

- Lors de la création d'une exception (le new), l'exception enregistre la pile des méthodes appelantes (fillInStackTrace())
- Lors du throw, l'exception descent la pile d'appels jusqu'à être attrapée par un catch
  - La VM fait aussi des throw (null, borne des tableaux, etc)
  - Si il n'y a pas de catch, la VM appel printStackTrace() sur l'exception après être remonté du main()

Types d'exceptions

Il existe 4 classes d'exceptions



Arbre de sous-typage des exceptions

2 Types d'exceptions

Le langage Java (pas la VM) oblige à attraper les sous-type de Throwable qui ne sont pas des sous-type de Error ou RuntimeException

- On peut soit
  - attraper (**try/catch**) l'exception ou
  - dire que la méthode appelante devra s'en occuper (**throws**)

Il y a donc deux types d'exceptions

- *checked*: le language oblige à les attraper
- *non-checked*: pas d'obligation

Exceptions levées par la VM

Les exceptions levées par la VM correspondent sont toutes unchecked

- Les Errors
  - erreurs de compilation ou de lancement
    - NoClassDefFoundError, ClassFormatError
  - problème de ressource
    - OutOfMemoryError, StackOverflowError
- Les RuntimeException
  - des erreurs de programmation
    - NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException

Exceptions et programmeur

Le programmeur va utiliser des exceptions pour assurer :

- Que son code est bien utilisé
  NullPointerException/IllegalArgumentException
- Que l'état de l'objet est bon
  IllegalStateException
- Que le code fait ce qu'il doit faire
  AssertionError

Il faut de plus gérer toutes les exceptions qui sont checked (IOException par ex.)

Exemple de pré-conditions

```
public class Stack {
  private final Object[] array;
  private int top;
  ...

  public void push(Object object) {
    Objects.requireNonNull(object);
    if (top >= array.length)
      throw new IllegalStateException("stack is full");
    array[top++] = object;
  }

  public Object pop() {
    if (top <= 0)
      throw new IllegalStateException("stack is empty");
    Object object = array[top];
    array[top--] = null;  // GC !
    return object;
  }
```

Pré-conditions

Et les commentaires ...

Le code **doit** avoir des commentaires de doc

```
public class Stack {
  /** This class implements a fixed size stack of objets.
   */
  public class Stack {

    /** Put the object on top of the stack.
     * @param value value to push in the stack.
     * @throws NullPointerException if the object is null
     * @throws IllegalStateException if the stack is full.
     */
    public void push(Object object) {
      ...
    }

    /** Remove the object from the top of the stack.
     * @return the value on top of the stack.
     * @throws IllegalStateException if the stack is empty.
     */
    public Object pop() {
```

Prog. par contrat

Il n'est pas rare de voir dans les codes plus compliqué en plus des pré-conditions

- Post-conditions sont utilisées pour :
  - vérifier que les opérations ont bien été effectués
    assert, AssertionError
- Invariants sont utilisées pour :
  - Vérifier que les invariants de l'algorithme sont préservés. assert, AssertionError

Exemple

```
public class Stack {
  private final Object[] array;
  private int top;
  ...

  public void push(Object object) {
    Objects.requireNonNull(object);
    if (top>=array.length)
      throw new IllegalStateException("stack is full");
    array[top++] = object;
    assert object == array[top - 1];
    assert top >= 0 && top <= array.length;
  }

  public Object pop() {
    if (top<=0)
      throw new IllegalStateException("stack is empty");
    Object object = array[top];
    array[top--] = null;  // GC !
    assert object == array[top + 1];
    assert top >= 0 && top <= array.length;
    return object;
  }
```

Pré-condition

Post-condition

Invariant

Le mot-clé **assert**

- Le mot-clé **assert** permet de s'assurer que la valeur d’une expression est vraie
- Deux syntaxes :
  - assert *test*; `assert i==j;`
  - assert *test* : *mesg*; `assert i==j:"i not equals to j";`
- Par défaut, les **assert** ne sont pas exécutés, il faut lancer **java -ea** (enable assert)

assert et **AssertionError**

Si le test booléen du **assert** est faux, la VM lève une exception **AssertionError**

```
public class NonNullList  {
  private final ArrayList<Object> list;
  ...
  public void add(Object o)  {
    //Objects.requireNonNull(o)      ooops !
    list.add(o);
  }
  public void transfer(Queue<Object> queue) {
    for(int i = 0; i < list.size(); i++) {
      Object o = list.get(i);
      assert o != null : a list element can't be null;
      queue.offer(o);
    }
  }
}
```

```
Exception in thread "main" java.lang.AssertionError: a list
element can't be null
    at NonNullList.transfer(NonNullList.java:15)
    at NonNullList.main(NonNullList.java:23)
```

Checked Exception

- Une checked Exception hérite de Throwable ou de Exception
- Le langage Java oblige
  - soit à attraper l'exception avec un try/catch
  - soit à indiquer que l'on propage celle-ci avec un throws
- On ne doit mettre un try/catch que si l'on peut faire quelque chose d'intelligent dans le catch
  - Au lieu de e.printStackTrace(), il faut utiliser throws !

Attraper une exception

try/catch permet d'attraper les exceptions

```
public static void main(String[] args) {
  int value;
  try {
    value=Integer.parseInt(args[0]);
  } catch(NumberFormatException e) {
    value=0;
  }
  System.out.println("value "+value);
}
```

pas obligatoire

parseInt

```
public static int parseInt(String s)
                throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, exc
character may be an ASCII minus sign `'-'` (`'\u002D'`) to indicate a negative value. The resulting integer value
as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` me

Parameters:

`s` - a `String` containing the `int` representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

`NumberFormatException` - if the string does not contain a parsable integer.

La directive throws

Indique qu'une exception peut-être levée dans le code mais que celui-ci ne la gère pas (pas de try/catch).

```
public static void f(String author) throws OhNoException {
  if ("dan brown".equals(author))
    throw new OhNoException("oh no");
}
public static void main(String[] args) {
  try {
    f(args[0]);
  } catch(OhNoException e) {
    tryToRecover();
  }
```

throws est nécessaire que pour les Throwable/ Exception (pas les Error ou les RuntimeException)

Multi-catch

On peut attraper plusieurs exceptions,
en les séparant par un |

```
public static void main(String[] args) {
  MethodHandle mh;
  try {
    mh = lookup().findStatic(Integer.class, "parseInt",
        methodType(int.class, String.class));
  } catch (NoSuchMethodException | IllegalAccessException e) {
    logger.warning(e.getMessage());
    System.exit(1);
    throw null;    // System.exit() peut ne pas fonctionner
  }
  System.out.println(mh);
}
```

Catch et initialisation de variable

Le compilateur vérifie tout les chemins possibles

```
public static void main(String[] args) {
  int value;
  try {
    value = Integer.parseInt(args[0]);
  } catch(ArrayIndexOutOfBoundsException e) {
    return;
  } catch(NumberFormatException e) {
    value = 0;
  }
  System.out.println("value "+value);
}
```

Sinon cela ne compile pas

Catch inatteignable

- Attention, les blocs **catch** sont testés dans l'ordre d'écriture !
- Un catch inatteignable est un erreur

```
public class CatchExceptionExample {
  public static void main(String[] args) {
    int value;
    try {
      value=...
    } catch(Exception e) {
      value=1;
    } catch(IOException e) { // jamais appelé
      value=0;
    }
    System.out.println("value "+value);
  }
}
```

Throw precis

Un catch sur un super-type est considérée comme une union des exceptions pouvant être levée

```
public MethodHandle getMH()
  throws NoSuchMethodException, IllegalAccessException {

  try {
    return lookup().findStatic(Integer.class, "parseInt",
        methodType(int.class, String.class));
  } catch (ReflectiveOperationException e) {
    logger.warning(e.getMessage());
    throw e;
  }
}
```

Le bloc finally

Sert à exécuter un code quoi qu'il arrive (fermer un fichier, une connection, libérer une ressources)

```
FileReader reader = new FileReader(args[0]);
try {
  doSomething(reader);
} finally {
  reader.close();
}
```

Le **catch** n'est pas obligatoire.

Finally et throw precis

Un block finally est traduit en recopiant le contenu à la fin du block try **et** dans un catch Throwable qui utilise le throw precis

```
    FileReader reader = new FileReader(args[0]);
    try {
      doSomething(reader);
      reader.close();
    } catch(Throwable e) {
      reader.close();
      throw e;
    }
```

Try avec une ressource

La syntaxe try() permet d'appeler le close() automatiquement à la fin du bloc.

```
try(FileReader reader = new FileReader(args[0])) {
  doSomething(reader);
}  // appel close() automatiquement ici
```

La variable doit être un sous-type d'AutoCloseable

Si l'appel à close() lève une exception alors que le bloc a déjà levé une exception, elle est stockée en tant d'exception supprimée (getSuppressed()) de l'exception déjà existante

Try avec plusieurs resources

La syntaxe try() permet utiliser plusieurs variables, les close() seront appelées dans l'ordre inverse des déclarations

```
try(FileReader r = new FileReader(args[0]);
    BufferedReader reader = new BufferedReader(r)) {
  doSomething();
}  // appel reader.close() puis r.close()
```

Ne pas attraper tout ce qui bouge

Comment passer des heures à débugger

```
public static void aRandomThing(String[] args) {
  return Integer.parseInt(args[-1]);
}
public static void main(String[] args) {
  ...
  try {
    aRandomThing(args);
  } catch(Throwable t) {
    // surtout ne rien faire sinon c'est pas drôle
  }
  ...
}
```

Eviter les catch(Throwable) ou catch(Exception), car on attrape aussi les runtimes (et les erreurs) !!

Alors **throws** ou **catch**

- Si l'on appel une méthode qui lève une exception non runtime
  - Catch si l'on peut reprendre sur l'erreur et faire quelque chose de cohérent sinon
  - Throws propage l'exception vers celui qui à appelé la méthode qui fera ce qu'il doit faire

Exception et StackTrace

- Lors de la création d'une exception, la VM calcule le *StackTrace*
- Le *StackTrace* correspond aux fonctions empilées (dans la pile) lors de la création
- Le calcul du *StackTrace* est quelque chose de couteux en performance.
- Les numéros de ligne ne sont affichées que compilé avec l’option debug
(pas par défaut avec ant)

Programmation par exception

- Déclencher une exception pour l'attraper juste après est très rarement performant
(la création du stacktrace coûte cher)

```
public class CatchExceptionExample {
  public static int sum1(int[] array) { // 5,4 ns
    int sum=0;
    for(int v:array)
      sum+=v;
    return sum;
  }
  public static int sum2(int[] array) { // 7,2 ns
    int sum=0;
    try {
      for(int i=0;;)
        sum+=array[i++];
    } catch(ArrayIndexOutOfBoundsException e) {
      return sum;
    }
  }
}
```

StrackTrace et optimisation

- Pour des questions de performance, il est possible de pré-créer une exception

```
public class StackTraceExample {
  public static void f() {
    throw new RuntimeException();
  }
  public static void g() {
    if (exception==null)
      exception=new RuntimeException();
    throw exception;
  }
  private static RuntimeException exception;
  public static void main(String[] args) {
    f(); // 100 000 appels, moyenne()=3104 ns
    g(); // 100 000 appels, moyenne()=168 ns
  }
}
```

Le chaînage des exceptions

Il est possible d'encapsuler une exception dans une autre

- **new Exception("msg", Throwable cause)**
- **new Exception("msg").initCause(Throwable cause)**

Permet de lever une exception assez générique en indiquant précisément l'exception ayant causée le problème en tant que cause.

Le chaînage des exceptions (2)

Comme on ne peut renvoyer que des IOException, on encapsule les SAXException dans des IOException

```
@Override
public void load() throws IOException {
  try {
    SAXParser parser = factory.newSAXParser();
    try(InputStream input = newInputStream(get("foo.xml"))) {
      parser.parse(input, new DefaultHandler() {
        @Override
        public void startElement(String uri, String localName,
            String qName, Attributes attributes) throws SAXException {
          ...
        }
      });
    }
  } catch(ParserConfigurationException | SAXException e) {
    throw new IOException(e);
  }
}
```

Le chaînage des exceptions (3)

Et si startElement lève une IOException, on l'encapsule puis on dé-encapsule

```
        @Override
        public void startElement(String uri, String localName,
            String qName, Attributes attributes) throws SAXException {
          String file = attributes.getValue("file");
          List<String> lines;
          try {
            lines = readAllLines(get(file), ISO_8859_1);
          } catch(IOException e) {
            throw (SAXException)new SAXException().initCause(e);
          }
        }
      });
    }
  } catch(ParserConfigurationException | SAXException e) {
    Throwable cause = e.getCause();
    if (cause instanceof IOException) {
      throw (IOException)cause;
    }
    throw new IOException(e);
  }
```