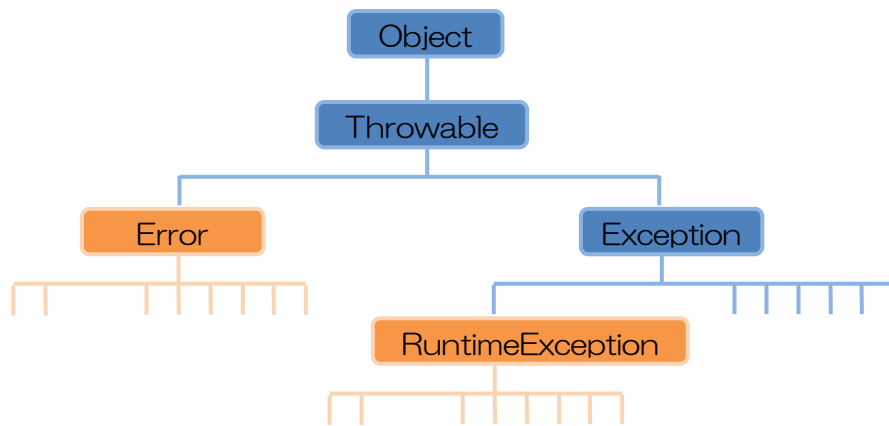


**例 外**  
例外とは

プログラムの実行中に発生するエラーです  
例外は、  
Throwable クラス、又はそのサブクラスのオブジェクト  
で表現されます

[例外クラスの階層]



例外クラスのクラス階層と分類

- |                                  |   |                                    |
|----------------------------------|---|------------------------------------|
| Error クラスと<br>そのサブクラス            | → | 続行不可能な例外<br>Virtual Machine のエラーなど |
| Exception クラスと<br>そのサブクラス        | → | 実行時に対処する必要のある例外<br>I/O エラーなど       |
| RuntimeException クラスと<br>そのサブクラス | → | 予め防げる例外<br>配列要素の数を超える代入など          |

例外クラスのタイプ

- |  |                       |
|--|-----------------------|
| <span style="display: inline-block; width: 15px; height: 15px; background-color: #4a7ebb; border: 1px solid black; margin-right: 5px;"></span> 以下 Checked 例外   | コンパイラは例外処理の約束をチェックします |
| <span style="display: inline-block; width: 15px; height: 15px; background-color: #e69d00; border: 1px solid black; margin-right: 5px;"></span> 以下 Unchecked 例外 | // チェックしません           |

※例外処理の約束については3ページ目を見てください

## 例外処理

例外処理とは 例外に対して行われる処理です

宣言 (try 文)

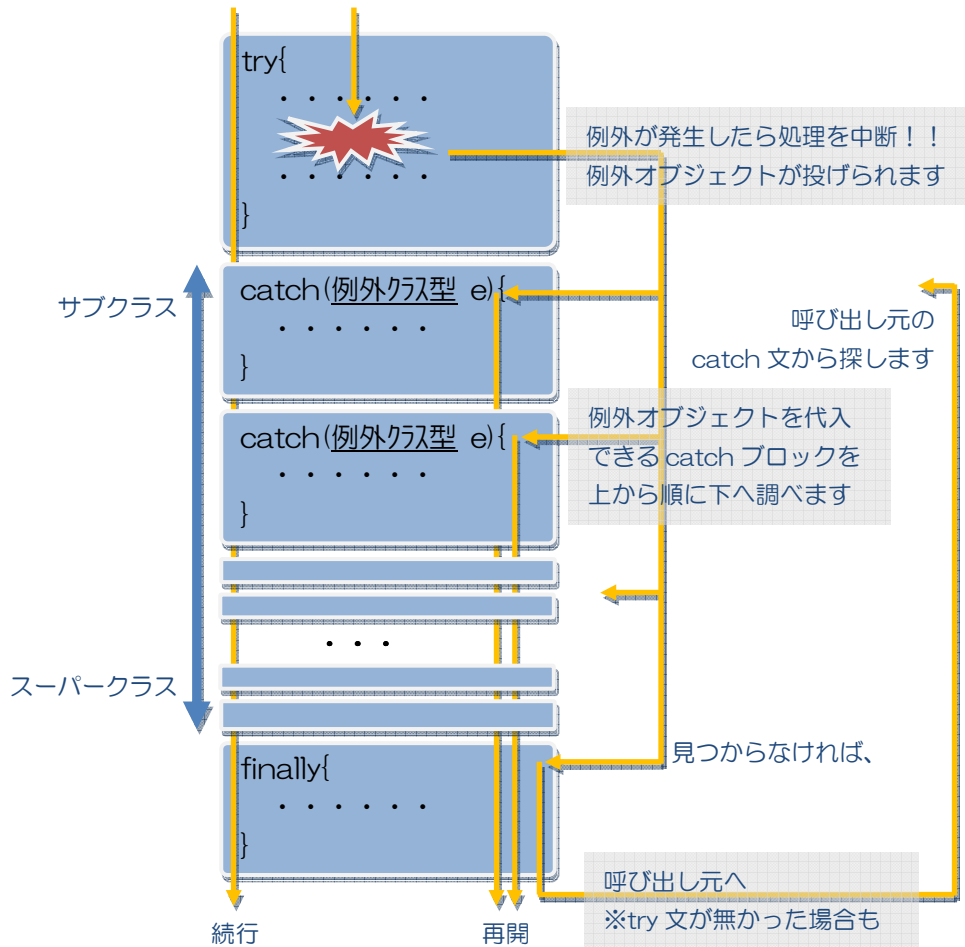
```
try{
  文
}catch(例外クラス型 1 e){
  文
}catch(例外クラス型 2 e){
  文
:
}finally{
  文
}
```

※1つのtry文は、1つのtryブロックと、少なくとも1つのcatchブロック又はfinallyブロックをもちます

機能

- ①try ブロック 例外の発生を監視
- ②catch ブロック 例外が発生した場合、例外に対応する処理を実行
- ③finally ブロック try ブロックに入ったら
  - ・例外の発生の有無に関わらず
  - ・break や return により try ブロックを出る場合 try 文を抜ける時に必ず実行

## 処理の流れ



## 例外の送付

例外の送付とは 例外を発生させることです

手順

1. Exception クラスを継承し各自の例外クラスを宣言します  
※一般に Throwable クラスを継承していればよいです  
また、既存の例外クラスを用いてもよいです
2. オブジェクトを生成します
3. throw オブジェクトを指す変数; により例外を発生させます

## throws 宣言

throws 宣言とは メソッドが例外を発生する可能性があることを明示します

例外の送付を行うメソッド、例外処理を記述していない例外や  
例外処理では catch していない例外を発生する可能性のあるメソッド  
にこれを宣言します

宣言

```
戻り値の型 メソッド名(引数リスト) throws 例外クラス1, 例外クラス2,...{  
    本体  
}
```

※例外クラス は例外に対応するクラスのスーパークラスでもよいです

## 例外処理の約束

あるメソッドA内で例外を送付するメソッドB（やコマンド）を実行する場合、  
基本的に次の

1. try 文によりメソッドA内でメソッドBの例外を処理してしまいます  
→ メソッドAの利用者は例外を気にせずにコードを作成できます
2. メソッドAに throws 宣言をします  
→ メソッドAの使用者にこのメソッドは例外が発生することを伝えます

のどちらかを行います

Checked 例外の場合 → コンパイラがこの約束をチェックします

例えば、  
IOException（キーボードなどの入力に関する例外）  
や Exception を継承した独自の例外クラスなど

Unchecked 例外の場合 → コンパイラはこの約束をチェックしません  
例外への対応はプログラマに任せられます

例えば、  
ArrayIndexOutOfBoundsException（配列範囲超え）  
ArithmeticException（ゼロでの除算）など

## throws 宣言をもつメソッドのオーバーライドの約束

オーバーライドの機能を妨げないように新しいメソッドの throws 宣言を次のようにします

オーバーライドする側のメソッドAは、オーバーライドされる側（スーパークラス又はインタフェース）のメソッドBが送出する例外のクラスとそのサブクラスを送出できます

理由は、オリジナルのメソッドBを使うプログラムでは、メソッドBが送出する例外に対応する例外処理しかないためです

### 継承の場合の例1：

```
void A() throws E1  
    ↓  
void A() throws E2
```

例外クラス E2 は、  
「例外クラス E1 と同じ、又はそのサブクラス」  
とします

### 継承の場合の例2：

```
void A() throws E1, E2  
    ↓  
void A() throws E3
```

例外クラス E3 は、  
「例外クラス E1 または E2 と同じ、又はそれらのサブクラス」  
とします

### 実装の場合の例：

```
void A() throws E3 ← ● void A() throws E1, E2
```

例外クラス E3 は、  
「例外クラス E1 または E2 と同じ、又はそれらのサブクラス」  
とします