OPEN STREAM

213-03.例外

Make IT your Success

目次

この講義について

目的：

- プログラムのエラーの種類について学び、例外がどこに位置するか、どのような状態を例外と呼ぶのかを知る。
- 例外のクラス階層について学ぶ。

ゴール：

try-catch-finallyを用いて例外をキャッチするプログラムを記述できる。

例外をスローするメソッドを作成できる。

エラー（不具合）の種類

プログラムのエラーを大別すると以下の３通り

①　作成したプログラムがプログラミング言語の文法に合っていない
②　プログラム実行時に想定した動作をするが、それ自体が良くない
③　プログラム実行時に想定していない動作をする

①はコンパイル時点でエラーとして検出される。
プログラマーがプログラムを修正して再度コンパイルすることで解消される。

②はいわゆる「仕様」の問題。
仕様・プログラムの再設計が必要。

この２つは設計・開発をしっかりやりましょう

③についてはプログラムの実行時に発生するもの。
プログラミング時点でエラー発生時の適切な処理を記述しておくことが出来る。
Javaでは「例外」という仕組みを使って実現する。

例外処理

プログラム実行中に予期しない事象が起きると「例外」が発生する。
(例外・・・Java中間コードを実行時に発生するエラー)

Javaでは本来行いたい処理で例外が発生する可能性がある場合、
それを呼び出した側のプログラムで適切に処理しなければならない。
→**例外処理**

例外の発生と例外クラスのオブジェクト(インスタンス)

例外の発生を、例外をスローする(例外を投げる)という。
スローされる例外は、例外クラスのオブジェクト(インスタンス)である。
例外のスローは、主に２通りあり、
プログラム実行時にJavaVMが自動的に行うか、
プログラム内にthrow文を記述してプログラムの処理として例外スローを行う。
例外処理は、スローされた(投げられた)例外(例外オブジェクト)を、キャッチ(変数に入れる)事で行う。

例外

例外処理の書き方：

```
try {
      // 本来行いたい処理
      //    ・・・  例外が発生する可能性がある範囲をtryで囲む

} catch( 例外クラス型 変数名 ) {
      // ↑catch句は複数書くことが出来る
      // (複数書く事で、異なる例外クラスを別々にcatchする事ができる)
      // catch句の引数で例外(例外クラスのインスタンス)をキャッチする

      // 例外が発生した時の処理
      //    ・・・  エラー発生時の適切な処理を書く

} finally {
      // 後始末を行う処理

}
```

catchとfinally

tryブロック内の処理で例外がスローされたときは、tryブロック内の処理はそこで終了し、自動的にtry句と同じ階層のcatch句の処理に遷移する。

catch句では、スローされた例外(例外オブジェクト、例外クラスのインスタンス)がcatchの引数で定義された例外クラス型と一致するかチェックされる。一致する場合、そのcatchブロックの処理が実行される。

例外は親クラス型でもcatch出来る。つまり、スローされた例外(例外クラスのインスタンス)の親クラス型がcatch文で記述されている場合も、そのcatchブロックの処理が実行される。

catch句は複数記述出来る。スローされた例外と一致するものがあるか上から順にチェックされる。

Tryブロックの処理が終わった後(例外が発生しない場合)、もしくは、catchブロックの処理が終わった後(例外が発生した場合)、finally句に遷移する。

finallyブロックはtryブロックに入ったら必ず処理される。

try with resources

try catch文のリソースの開放を自動化する

例文

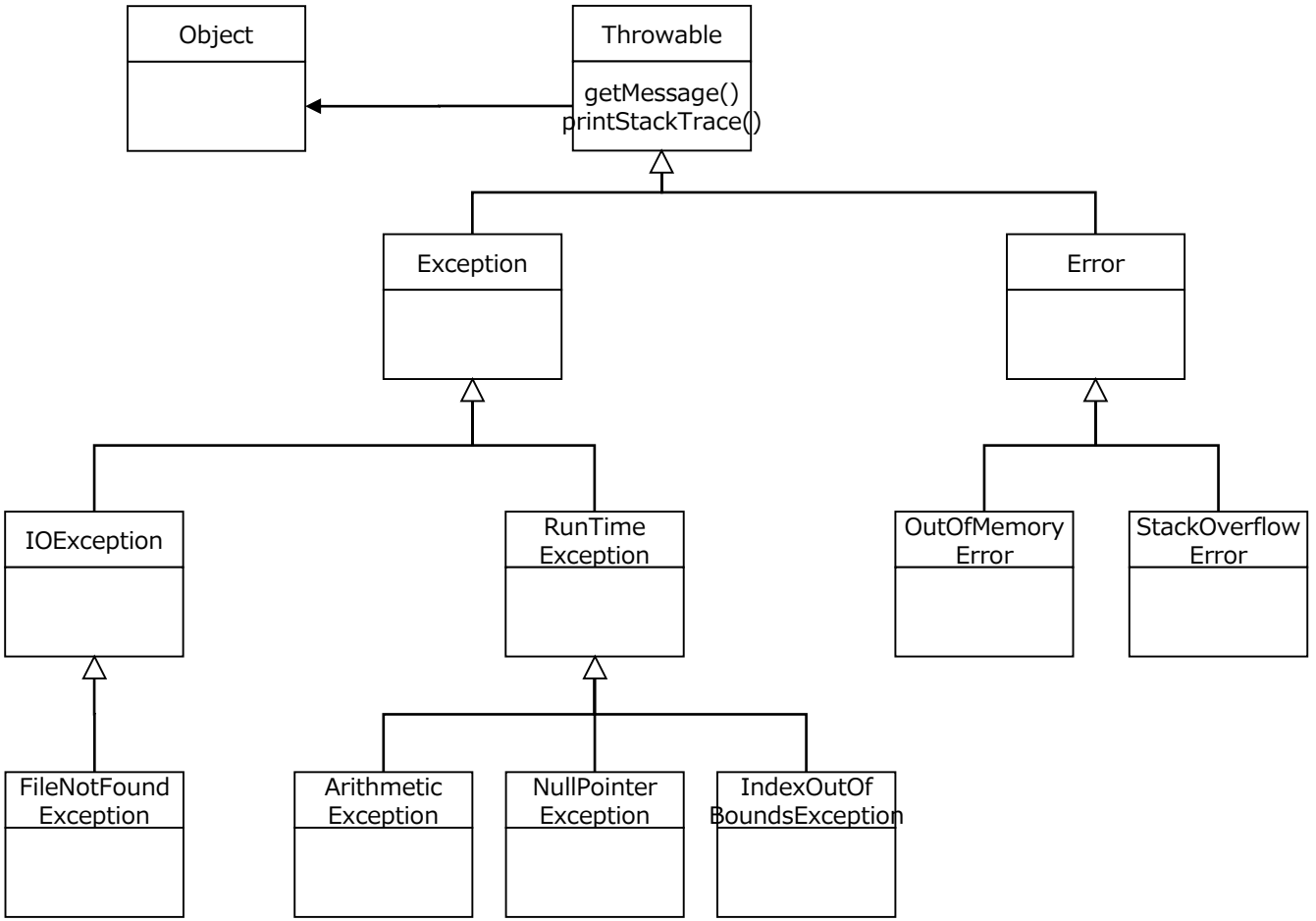
```
try (Connection conn = ds.getConnection()) {
   // DB処理など、リソースを使用した処理

} catch (SQLException e) {
   // 例外処理

}
```

インタフェース java.lang.AutoCloseable は close()メソッドを持っている。
AutoCloseableを実装しているクラスはtry with resourceステートメントを使う事ができる。
tryブロックを実行し終えるとclose()メソッドがリソースを解放するために呼び出される。
※catchブロックに入る場合、tryブロックを抜ける時にリソースが開放される。
※クローズされる順番は定義順の逆。
※catchブロック, finallyブロックの前にクローズ処理が行われる。

例外クラスのクラス階層



例外クラスのクラス階層

Throwable

Throwableクラスはすべてのエラー、例外のスーパークラス。

通常、Throwableクラスをプログラム内で直接記述することは無い。

Error

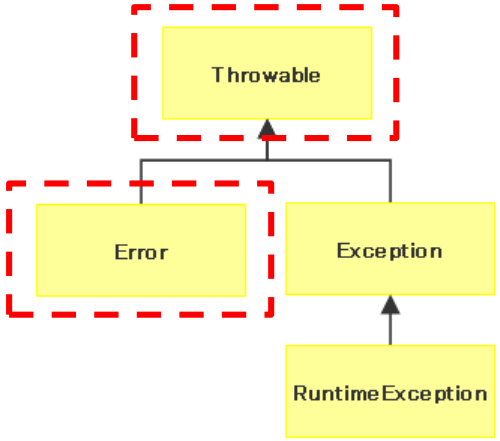
Errorクラスは、本来アプリケーションで発生してはならない重大なエラーを表現するクラス。

Errorクラスはプログラム上で明示的にスローすることは無い。スローされたとしてもキャッチすべきではない。

また、プログラム内で拡張したり、サブクラスを定義することは無い。

主なサブクラス

- OutOfMemoryError
- StackOverflowError
- NoClassDefFoundError



したがって、プログラマが設計上考慮する必要のある例外は
ExceptionクラスとRuntimeExceptionになる。

例外

もしErrorクラスの子クラスの例外が発生したらどうするか？

・プログラム全体の構成・設計を見直す。

・プログラムの実行環境に問題ないか見直す。

Javaでは例外(エラー)を次のように区別している

- 例外 Exception … アプリケーション内で復帰可能
- エラー Error … アプリケーション内で復帰不可能
  - ・プログラムのコードを書き直す
  - ・環境設定を見直す

そのなかで覚えるべきError

- StackOverflowError：スタック領域がない
- OutOfMemoryError：ヒープ領域がない

例外クラスのクラス階層

Exception

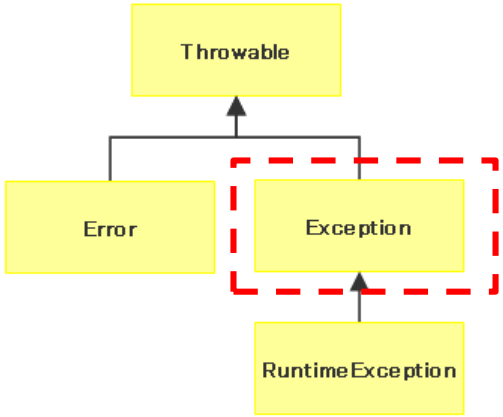
Exceptionクラスは、アプリケーションの一般的な例外的状況を表現する。

Exceptionクラスをそのまま使用することはなく、例外の状況に応じて適切なサブクラスを使用する。

Exceptionクラスを継承し、独自の例外を作成する場合もある。

主なサブクラス

- ClassNotFoundException
- FileNotFoundException
- CloneNotSupportedException
- DataFormatException



Exceptionクラスの子クラスについて

必ず呼び出し側のプログラムで適切に処理しなければならない。

例外処理を書かないとコンパイルエラーとなる。

強制することでエラー時に適切に処理するように促している。

例外クラスのクラス階層

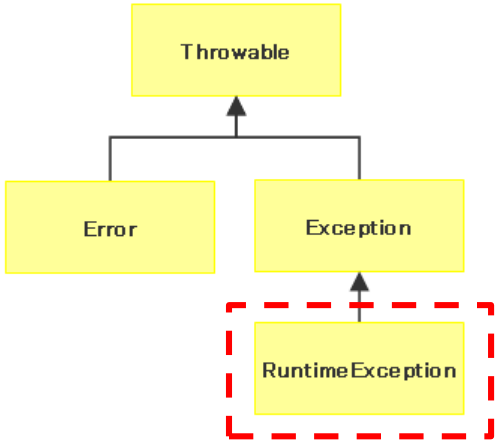
RuntimeExcption

RuntimeExceptionはExceptionのサブクラスだが、一般的にはコンパイルエラーとして検出しきれないプログラムのコーディングミスに伴い、実行時にスローされる例外クラス。

状況に応じてRuntimeExceptionを継承し、独自の例外を作成することがある。

主な子クラス(サブクラス)

- ClassCastException
- IllegalArgumentException
- NullPointerException
- IndexOutOfBoundsException



RuntimeExceptionクラスの子クラスについて

呼び出し側のプログラムでの処理は任意となっている。

例外処理を書かなくてもコンパイル時にエラーとならない。

（通常はプログラムをしっかり設計・開発することで
この例外は発生させないようにする）

上位（呼び出し元）に「適切な処理」をまかせたい時

「適切な処理」は難しく、それを書くのがふさわしい場所がある。
上位（呼び出し元）に任せるようにすることが出来る。

書き方：

```
メソッド名(引数) throws 発生する可能性のある例外クラス名, .... {
```

呼び出した側は「発生する可能性のある例外」の処理を記述しなければならない。
必ずプログラム全体のどこかで例外を処理しなければならない。

例外のthrow

また例外を発生させることも出来る。

書き方：

```
throw 例外クラス型の変数;
```

例外クラスの変数を宣言してインスタンスを生成するのを忘れずに

書き方：

```
例外クラス型　変数名
変数名　＝ new 例外クラス名(引数);
```

引数は例外クラスのコンストラクタを参照

例外とスタック

例外がスローされると例外クラス型の変数に例外の内容が保持されている。
どういうエラーがどこで発生したのかが分かる貴重な情報。

スタックで説明したように、メソッドが実行されると、そのメソッドのアドレスもスタックに入る。
ということは、スタックを見るとどのように呼び出されて来たかを確認することができる。

例外クラス型にはスタックを確認出来るメンバ（メソッド・フィールド）が用意されている。

例：

printStackTraceメソッド

例外クラスを自作する

例外クラス（通常Exceptionクラス）を継承することで例外クラスを自作できる。

どのような場面で自作の例外クラスを作るのか？

・業務的な意味合い

・そのシステム独自の意味合い