

第25章 例外と例外処理

目次

- 例外とは
- 例外クラス
- 例外処理
- 例外を投げる

例外とエラー

例外:

プログラムの処理で対処可能

エラー:

プログラム実行中にJVMでは対処できない
(プログラムの処理で対処できない)

チェック例外と非チェック例外

チェック例外:コンパイラが発生を予測できる例外

→事前に、その例外が発生した場合に対処となる
処理の記述が必要。

処理を記述しないと、コンパイルエラーとなる。

(例)

- 「標準入力を行う際に入力処理に失敗した」
- 「通信先の相手が応答しない」

チェック例外と非チェック例外

非チェック例外:コンパイラが発見できない例外
→発生する場所が予測できないため、
基本的に対処となる処理は記述不要。

(例)

- 「存在しない配列要素へのアクセス」
- 「null状態変数へのオブジェクト参照」

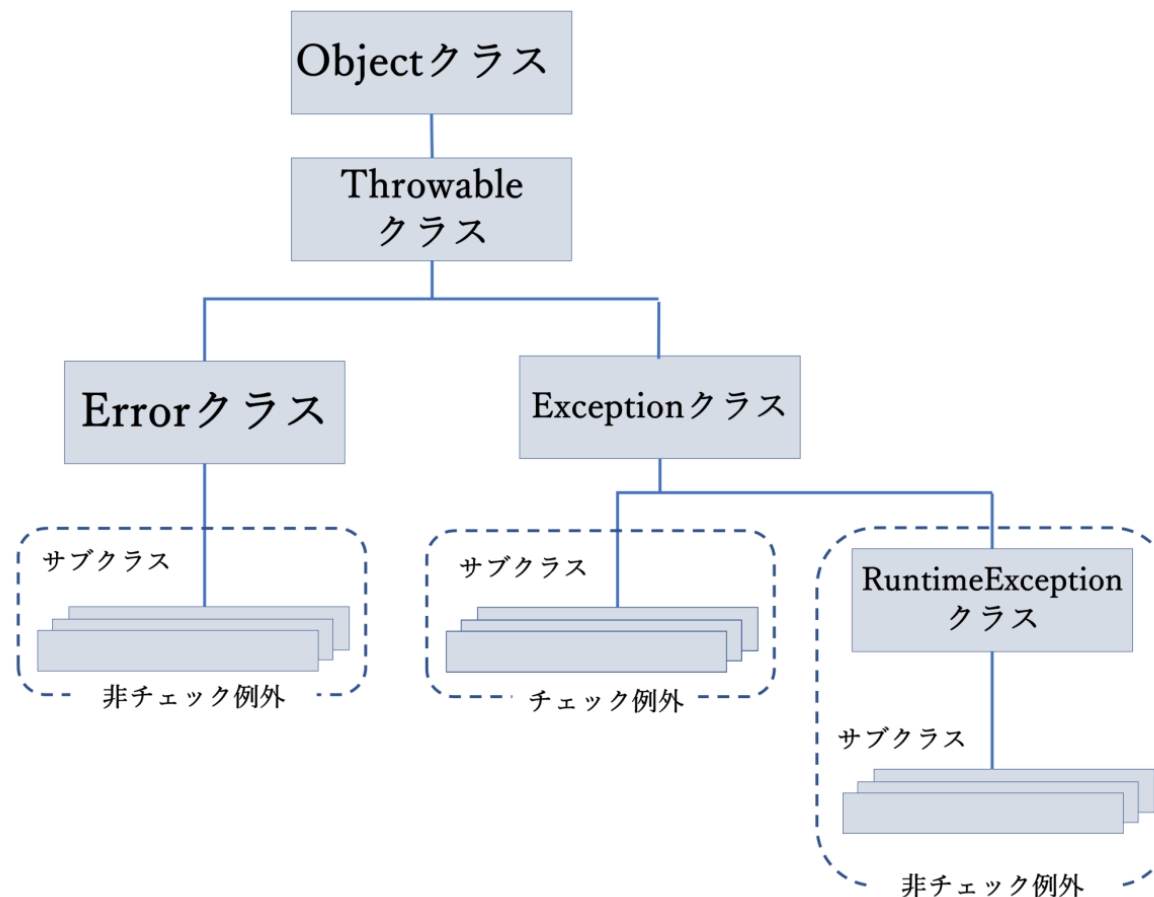
例外クラス

例外の情報をクラスとして扱う。

これらのクラスを「例外クラス」という。

標準的な例外クラスは、クラスライブラリとして提供されている。

例外クラスの継承関係



例外クラスの継承関係

■ Throwableクラス

Java言語のすべてのエラーと例外のスーパークラス。
ThrowableクラスのサブクラスとしてErrorクラスと
Exceptionクラスが提供されている

■ Errorクラス

エラーに関するクラス。エラーをプログラムで対応するのは困難。
そのため、このクラスおよびサブクラスへの対処となる処理は
実装する必要なし。

例外クラスの継承関係

■ Exceptionクラス

例外全般を扱うクラス。対処となる処理を記述すべきかどうかは、(一部を除いて)発生する可能性のある例外クラスがチェック例外であるかどうかで決まる。

■ RuntimeExceptionクラス

すべての非チェック例外のスーパークラス。
非チェック例外はプログラミング中に度々遭遇する。
非チェック例外については、代表的な例外クラスだけでも、名前と発生した原因を把握しておくとい。

Errorクラスのサブクラス

→非チェック例外(例外処理は不要)

クラス名	意味	説明
VirtualMachineError	JVMの異常	※テキスト「Java下巻 P268」を参照
OutOfMemoryError	メモリ不足によるエラー	
StackOverflowError	不十分なJavaスタックによるエラー	
NoClassDefFoundError	クラスパスの問題によるエラー	

※詳細は教科書中の表を参照。

RuntimeExceptionクラスのサブクラス →非チェック例外(例外処理は必須ではない※)

クラス名	意味	説明
ClassCastException	キャストの失敗	※テキスト「Java下巻 P268～269」を参照
IllegalArgumentException	不正な引数	
NumberFormatException	数値形式文字列の書き誤り	
IllegalStateException	不正な状態での呼び出し	
IndexOutOfBoundsException	不正な状態での呼び出し	
ArrayIndexOutOfBoundsException	配列の要素番号の範囲外指定	
NullPointerException	使ったオブジェクトがnull	
ArithmeticException	算術エラー	

※詳細は教科書中の表を参照。

RuntimeExceptionクラス以外のExceptionのサブクラス →チェック例外（例外処理は必須）

クラス名	意味	説明
IOException	入出力処理の失敗	※テキスト「Java下巻 P269」を参照
FileNotFoundException	ファイル関係の例外	
ClassNotFoundException	クラス定義やメソッドの呼び出し関係の問題	

※詳細は教科書中の表を参照。

【Sample2501 実行時例外の発生】を作成しましょう



Sample2501のポイント

サンプルコードでは、「0で割る」ことで
非チェック例外を起こしている。

0での割り算は無限大になるので、int型では処理できず
例外となり、コンソールには例外発生時の状況が表示される。

Exception in thread "main" java.lang.ArithmeticException: /
by zero

0による割り算

パッケージ名

クラス名

Sample2501のポイント

例外の発生場所(スタックトレース)

例外が送出された行と、それを呼び出したメソッドの記述がある行が表示される。その内容を読み解くことで、例外の原因を特定できる。

```
at exception.Sample2501.calc(Sample2501.java:10)
at exception.Sample2501.main(Sample2501.java:5)
```

← 例外の発生した行

← calcメソッドを
呼び出した行

Sample2501のポイント

例外の発生した行

```
at exception.Sample2501.calc(Sample2501.java:10)
```



```
exceptionパッケージの/Sample2501クラスの  
/calcメソッド/( Sample2501.javaファイルの10行目)
```


Sample2501のポイント

calcメソッドを呼び出した行とそのメソッド

at exception.Sample2501.main(Sample2501.java:5)



exceptionパッケージの/Sample2501クラスの
/mainメソッド/(Sample2501.javaファイルの5行目)

例外処理とは

例外が発生した場合の対処となる処理のこと。

例外処理を行わずに例外が発生した場合、
プログラムは強制的に処理が中断する。

例外処理が行われれば処理が中断せず、続行する。

try-catch-finally文

例外処理はtry-catch-finally文を利用して実装できる。

例外処理を実装すると、例外が発生した際に、
その例外の情報を捕捉し(キャッチし)、
例外の種類に応じて特定の処理を実行する。

try-catch-finally文

try-catch-finally文はtryブロック、catchブロック、finallyブロックから構成されている。

- tryブロック
例外が発生しそうな箇所を囲む。
- catchブロック
例外が発生した時の処理をブロック内に定義する。
- finallyブロック
例外の有無に関わらず必ず実行したい処理をfinallyブロックに定義する。

try-catch-finally文

構文

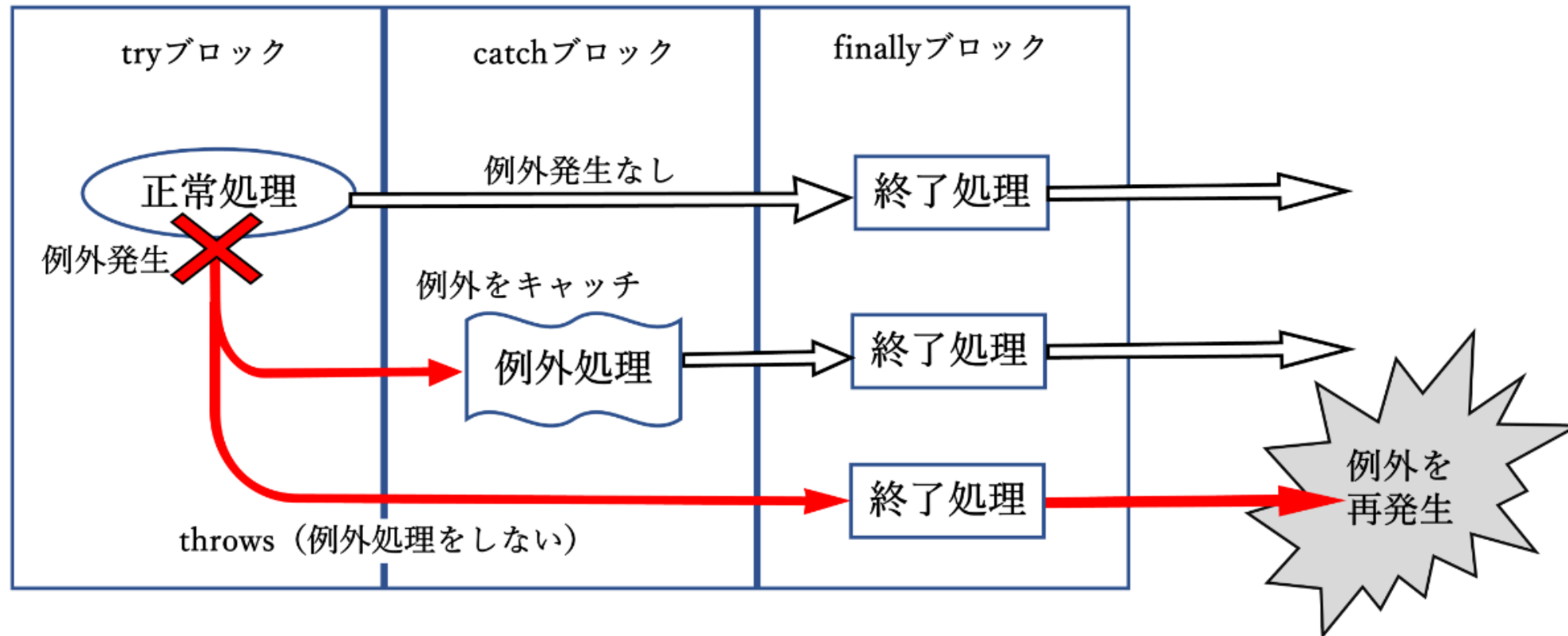
```
try {  
    例外が発生する可能性のある処理  
} catch (例外クラス名 変数名){  
    例外発生時の処理  
} finally {  
    例外の有無に関わらず行う処理  
}
```

try-catch-finally文

try-catch-finally文の
すべてのブロックを記述する必要はない。
try-catch-finally文のブロックは
下記の組み合わせで記述可能。

- try-catch
 - try-finally
 - try-catch-finally
- ※tryブロックのみの定義はコンパイルエラー

try-catch-finally文



【Sample2502 try-catch-finally文】 を作成しましょう



Sample2502のポイント

Sample2502.javaでは、Sample2501の処理に
例外処理(try-catch-finally)を追加している。
「0で割る」処理が実行されたタイミングで例外
(ArithmeticException)が発生する。

```
try {  
    calc();  
} catch (ArithmeticException e) {  
    System.out.println("計算中にエラーが発生しました。  
    ");  
} finally {  
    System.out.println("終了しました。");  
}
```

Sample2502のポイント

例外が発生するとtryブロック内の処理は中断され、catchブロック内の処理に移る。

```
try {  
    ← 例外発生 ← calc();  
} catch (ArithmeticException e) {  
    System.out.println("計算中にエラーが発生しました。");  
} finally {  
    System.out.println("終了しました。");  
}
```

Sample2502のポイント

実行結果を見るとfinallyブロック内の処理も実行されている。

計算中にエラーが発生しました。
終了しました。

```
} finally {  
    System.out.println("終了しました。");  
}
```



Sample2502のポイント

finallyブロックは、例外が発生してなくても、必ず行いたい処理があれば利用する。

(代表的な処理)

リソースの開放

(データベースのclose処理やファイルのclose処理など)

リソース:

プログラム処理中での利用が済んだら

必ず閉じられなければならないオブジェクトのこと。

複数のcatchブロックを使用した例外処理

tryブロック内の処理で複数種類の例外が発生する可能性がある場合、例外ごとにcatchブロックを記述できる。

```
} catch (例外クラス名1 変数名1) {  
    例外発生時の処理1  
}  
} catch (例外クラス名2 変数名2) {  
    例外発生時の処理2  
}  
}
```

複数のcatchブロックを使用した例外処理


注意点

- 上から順番に「()」内に指定された例外と照合を行う
- 例外クラスを指定すると、そのサブクラスの例外もキャッチする対象となる

複数のcatchブロックを使用した例外処理

Exceptionクラスを指定すると、
すべての例外を一括してキャッチする。
下記のサンプルコードでは、
2つ目のcatchブロックは実行されない(到達不可能)。
コンパイルエラーとなる。

```
} catch(Exception e) {  
} catch(IOException e) {  
}
```



すべての例外がキャッチされる

到達不可能

【Sample2503 複数のcatchブロック】 を作成しましょう

Let's try!



Sample2503のポイント

実行結果は1つ目のcatchブロックの
例外が発生した時の処理。

インデックスを指定して配列の値を参照する際、
存在しない要素を参照した場合に例外が発生する。

一人目の名前は田中さんです。
二人目の名前は鈴木さんです。
java.lang.ArrayIndexOutOfBoundsException: 2
配列のインデックスが不正です。
at lesson25.Sample2503.main(Sample2503.java:20)

Sample2503のポイント

2つ目のcatchブロックはIOException
(入出力処理に失敗した場合に発生)する例外クラス。
InputStreamやBufferedReaderは、
データ入力を抽象化したもの。
データを読み取る際は常にIOExceptionをキャッチする
コードを書く必要がある。

throws文

例外が発生する可能性のあるメソッドには、
呼び出し元へその例外の情報を渡すことの明示が必要。
例外の情報を渡すにはthrows文を使用する。

修飾子 戻り値の型 メソッド名(引数リスト) throws 例外クラス名 {



メソッド内で発生する可能性のある例外のクラス名を記述する

throws文

チェック例外が発生する可能性があるメソッド:

→必ず記述

非チェック例外が発生する可能性があるメソッド:

→記述は任意

throws文

throws文を記述すると、指定された例外が発生したとき、その例外の情報(例外クラスのオブジェクト)が呼び出し元のメソッドに渡される。

throws文

指定したい例外が複数ある場合は、
「,(カンマ)」区切りで記述する。

```
void readFile() throws FileNotFoundException, IOException {  
    ...  
}
```



FileNotFoundException、またはIOExceptionが発生したとき、
呼び出し元にその例外の情報が渡される

throws文

呼び出し元のメソッドは、例外の情報を受け取ると、下記のいずれかの処理を実行する必要がある。

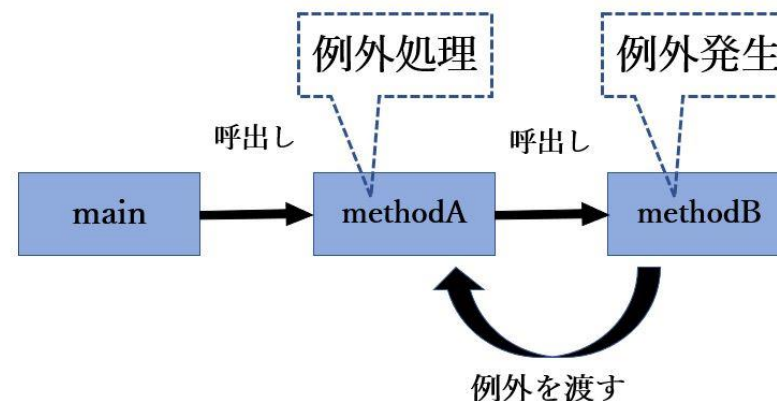
- catchブロックでキャッチして、例外処理を実行する。
- throws文を記述して、呼び出し元のメソッドに例外の情報を渡す。

throws文

(例)

main()メソッド→methodA()メソッド→methodB()メソッドの
順に呼び出し

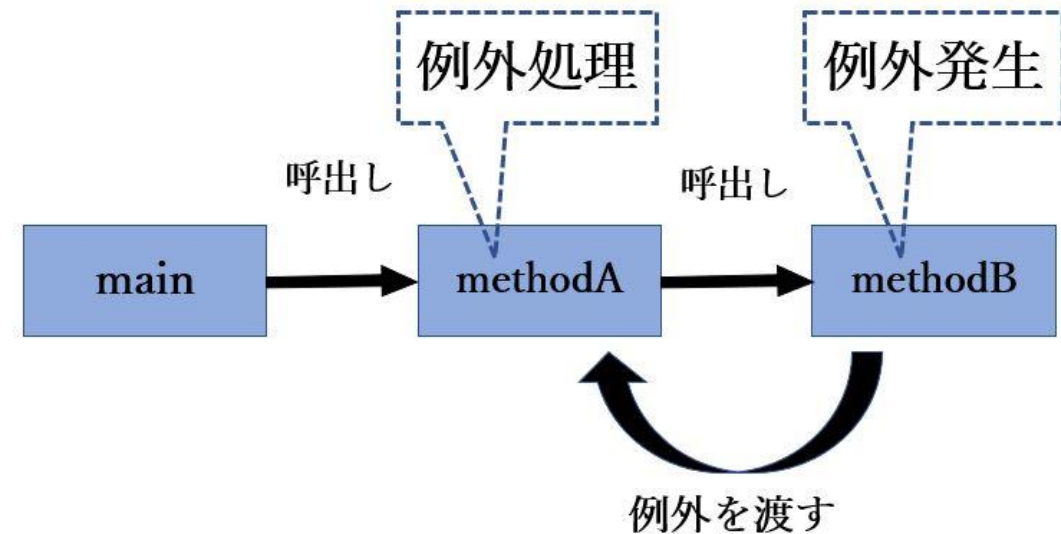
methodB()メソッドにはthrows文が、
methodA()メソッドには例外処理が記述されている



throws文

methodB()メソッドで例外が発生すると、処理が中断され、例外の情報がmethodA()メソッドに渡される。

methodA()メソッドは例外の情報を受け取ると、例外処理を実行し、そのまま処理が続行される。



throws文

throws文を活用すると、意図した階層のメソッドで例外処理を実行できる。

main()メソッドにもthrows文は記述できる。
その場合はJVMへ伝達されるため、
例外処理をしないのと同じことになる。

【Sample2504 throws文の利用】を作成しましょう



Sample2504のポイント

例外が発生した際に、InputName()メソッドで発生した例外を、main()メソッドにthrowsで渡している。
main()メソッドで例外処理を行ってる。

```
public static void inputName() throws IOException,  
ArrayIndexOutOfBoundsException {
```

```
try {  
    inputName();  
} catch
```

Sample2504のポイント

main()メソッド内ではマルチキャッチ(複数の例外をまとめてキャッチするための構文)で例外をキャッチしている。

マルチキャッチをするには、
各例外クラスを「| (パイプライン)」で区切る。

```
catch (ArrayIndexOutOfBoundsException | IOException e) {  
}
```

Sample2504のポイント

マルチキャッチは、どちらの例外がキャッチされたかが分かりにくいというデメリットがある。

そのため、上記のサンプルコードではブロック内に「`e.printStackTrace();`」(発生した例外をコンソール上に出力するメソッド)を記述して、どの例外が発生したか確かめられるようにしている。

```
catch (ArrayIndexOutOfBoundsException | IOException e) {  
    e.printStackTrace();  
}
```

例外オブジェクトで利用できる主な取得メソッド

メソッド	取得できる情報
<code>getCause()</code>	例外の原因
<code>getMessage()</code>	例外メッセージ
<code>getStackTrace()</code>	スタックトレース
<code>printStackTrace()</code>	スタックトレース(標準出力に出力)

throw文

意図的に例外を発生させる処理のこと。
引数の値や入力した文字列が想定外の値だった場合に
意図的にメソッドの処理を中断したいときなどに利用する。

throw 例外クラスのオブジェクト

例外クラスから生成されたオブジェクトの情報(参照)を
throw文の後ろに記述することで、対象の例外を発生させる。
throw文により例外を発生させることを
「例外を投げる(送出する)」いう。

throw文

チェック例外を投げる場合、throw文に加えて、throws宣言を記述する。

```
void methodA throws IOException{  
    if (a == null) {  
        throw new IOException();  
    }  
}
```

throw文

非チェック例外を投げる場合は、throws宣言の記述は不要。

```
void methodB {  
    if (a == null) {  
        throw new NullPointerException();  
    }  
}
```

【Sample2505 throw文】を作成しましょう



Sample2505のポイント

入力された文字列が設定したNGワードと一致した場合、
throw文で例外Exceptionを発生させる。

呼び出し元であるmain()メソッドではExceptionクラスに対して
throws宣言を記述しているため、例外処理は行われず、
スタックトレースが出力される。

好きな単語を入力してください>ITSchool

Exception in thread "main" java.lang.Exception: 入力された内
容がNGワードです

at lesson25.WordChecker2505.checkNGWord(WordChecker2505.java:7)

at lesson25.Sample2505.main(Sample2505.java:15)

Sample2505のポイント

文字列が一致しない場合は、正常処理の実行結果が
コンソールに出力される。

好きな単語を入力してください> 入力した単語
問題ありません。

独自例外クラス

プログラマが独自の例外クラスを定義することもできる。
一般的には、Exceptionクラスを継承した
publicなクラスとして定義する。

```
修飾子 class クラス名 extends Exception{
```

Exceptionクラスを継承することで、Exceptionクラスおよび
そのスーパークラスであるThrowableクラスが提供している
メソッドを引き継げる。

独自例外クラス

クラスブロック内には特にメンバを記述する必要はない。

```
public class MyException extends Exception{
```

独自例外クラスを定義した後は、
標準的な例外クラスと同じように扱える
ただし、独自例外クラスはthrow文を使用しないと発生しない。
プログラマが意図したタイミングで発生させるように
実装する必要がある。

【Sample2506 独自例外クラス】を作成しましょう



章のまとめ

- 想定内の不具合を例外、
想定外の致命的な不具合をエラーと言います。
- 処理が必須な例外をチェック例外と言い、
処理が任意の例外を非チェック例外といいます。
- 例外クラスは継承関係で構成されていて、幾つかの
スーパークラスは、その特徴を覚えておく必要があります。
- 例外はtry-catch-finallyを使って処理します。
- 呼び出し元のメソッドに対して例外を投げるには
throws宣言を使います。
- 意図的に例外を起こすにはthrow文を使います。