

12 例外



Created by GT F

Last updated 2019-08-18

- [例外処理](#)
- [マルチ例外処理](#)
- [例外をスロー](#)
- [カスタマイズ例外](#)
- [代表的な例外](#)
- [実戦](#)

例外処理

- `try` は例外的な事態が発生する可能性がある操作を行うときのことを示します。
- `catch` は、このコードブロックが例外な事態に対処することを示します。
- `finally` 例外が発生しても**発生しなくても**実行される処理です。
- `finally` は省略可能です。
- すべての例外クラスは `Exception` クラスを継承しています。

以下例外処理の構文です。

```
1 try {
2     // 例外的な事態が発生する可能性がある処理ブロック
3 } catch (Exception e) {
4     // 例外な事態に対処する
5 } finally {
6     // 最終に必ず実施すること。
7 }
```

実例 1 : ArithmeticException

```
1 public static void main(String[] args) {
2     System.out.println(20 / 0);
3 }
```

上記コードを実施すると、以下エラーが発生します。

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

実例 2 : ArithmeticExceptionを処理する

```
1 public static void main(String[] args) {
2     try {
3         System.out.println(20 / 0);
4     } catch (ArithmeticException e) {
5         System.out.println("エラーが発生します");
6     }
7 }
```

```
6     }
7 }
```

なぜ例外処理：エラー処理ない場合、例外発生する時、プログラムが直接に終了させた。後続処理を続けません。

マルチ例外処理

1つの `try` 文から複数の例外を処理する可能です。

```
1 public static void main(String[] args) {
2     try {
3         // コード
4     } catch (ArithmeticException e) {
5         System.out.println("算術例外");
6     } catch (NullPointerException e) {
7         System.out.println("NullPointerException");
8     }
9 }
```

複数の `catch` 文ではない、演算子 `|` 利用も可能です。

```
1 public static void main(String[] args) {
2     try {
3         // コード
4     } catch (ArithmeticException | NullPointerException e) {
5         System.out.println("算術例外 or NullPointerException");
6     }
7 }
```

なお、`catch` 文は例外優先順を注意してください。

```
1 public static void main(String[] args) {
2     try {
3         // コード
4     } catch (Exception | NullPointerException e) {
5         System.out.println("...");
6     }
7 }
```

`Exception` はすべて例外クラスの親クラスである為、該当 `catch` 文はあまりよくないです。

例外をスロー

例外発生する処理は絶対メソッドである為、例外の処理はメソッドの呼び出す元にしないといけません。呼び出す元を処理したくない時は `throws` する必要があります。

例 1：例外を自分（`readFile`）で処理する場合

```
1 public class Kicker {
2
```

```

3      public static void main(String... args) {
4          readFile(new File("data.txt"));
5      }
6      // メソッド自体に例外を処理する
7      public static void readFile(File f) {
8          FileInputStream in = null;
9          try {
10             in = new FileInputStream(f);
11             in.close();
12         } catch (FileNotFoundException ex) {
13             Logger.getLogger(Kicker.class.getName()).log(Level.SEVERE, null, ex);
14         } finally {
15             try {
16                 in.close();
17             } catch (IOException ex) {
18                 Logger.getLogger(Kicker.class.getName()).log(Level.SEVERE, null, ex);
19             }
20         }
21     }
22 }

```

例 2 : 例外を自分 (readFile) で処理したくない時

```

1  public class Kicker {
2
3      public static void main(String... args) {
4          try {
5              readFile(new File("data.txt"));
6          } catch (IOException ex) {
7              Logger.getLogger(Kicker.class.getName()).log(Level.SEVERE, null, ex);
8          }
9      }
10     // throwsキーワードを追加、該当メソッドを呼び出す元に例外処理を依頼する
11     public static void readFile(File f) throws FileNotFoundException, IOException {
12         FileInputStream in = new FileInputStream(f);
13         in.close();
14     }
15 }

```

カスタマイズ例外

例外をカスタマイズ可能です。すべての例外は親クラスException継承する必要があります。

```


1  public class Kicker {
2      // カスタマイズ例外クラスは親クラスExceptionを継承する必要
3      public static class MyException extends Exception {
4          public String message;
5          public MyException(String msg) {
6              this.message = msg;
7          }
8      }
9
10     public static void dosomething(String input) throws MyException {

```

```

11         if (input.equals("123")) {
12             // 例外を throwする
13             throw new MyException("カスタマイズ例外");
14         }
15     }
16
17     public static void main(String... args) {
18         try {
19             dosomething("123");
20         } catch (MyException ex) {
21             Logger.getLogger(Kicker.class.getName()).log(Level.SEVERE, null, ex);
22         }
23     }
24
25 }

```

 キーワード throw と キーワード throws を注意してください。

代表的な例外

クラス名	説明
ClassNotFoundException	クラスが見つからない
RuntimeException	実行時に例外が発生
IOException	入出力の例外が発生
FileNotFoundException	ファイルが見つからない
NumberFormatException	不適切な文字列を数値に変換
ArrayIndexOutOfBoundsException	配列の範囲外を指定
NullPointerException	nullオブジェクトにアクセス

実戦

質問 1 : 配列の範囲外を指定している場合、ArrayIndexOutOfBoundsException例外が発生します。該当例外を処理するサンプルコードを書いてください。

質問 2 : Validation Exceptionを作成して、ユーザー入力した値を検証します。該当は実際商用アプリで利用しています。

STEP1 : ValidationException クラスを定義する

```
1 public class ValidationException extends Exception {  
2     private String name; // カラム名称  
3     private String message; // エラーメッセージ  
4 }
```

STEP2 : チェック処理（サンプル）

```
1 if(name.matches("[0-9]+$")) {  
2     throw new ValidationException("名称", "半角数字を入力してください。");  
3 }
```

 Like Be the first to like this

No labels 