LightHouse Academy

2.5 例外

- 例外
- Java の例外
- 例外処理

www.lighthouseit.academy

目次

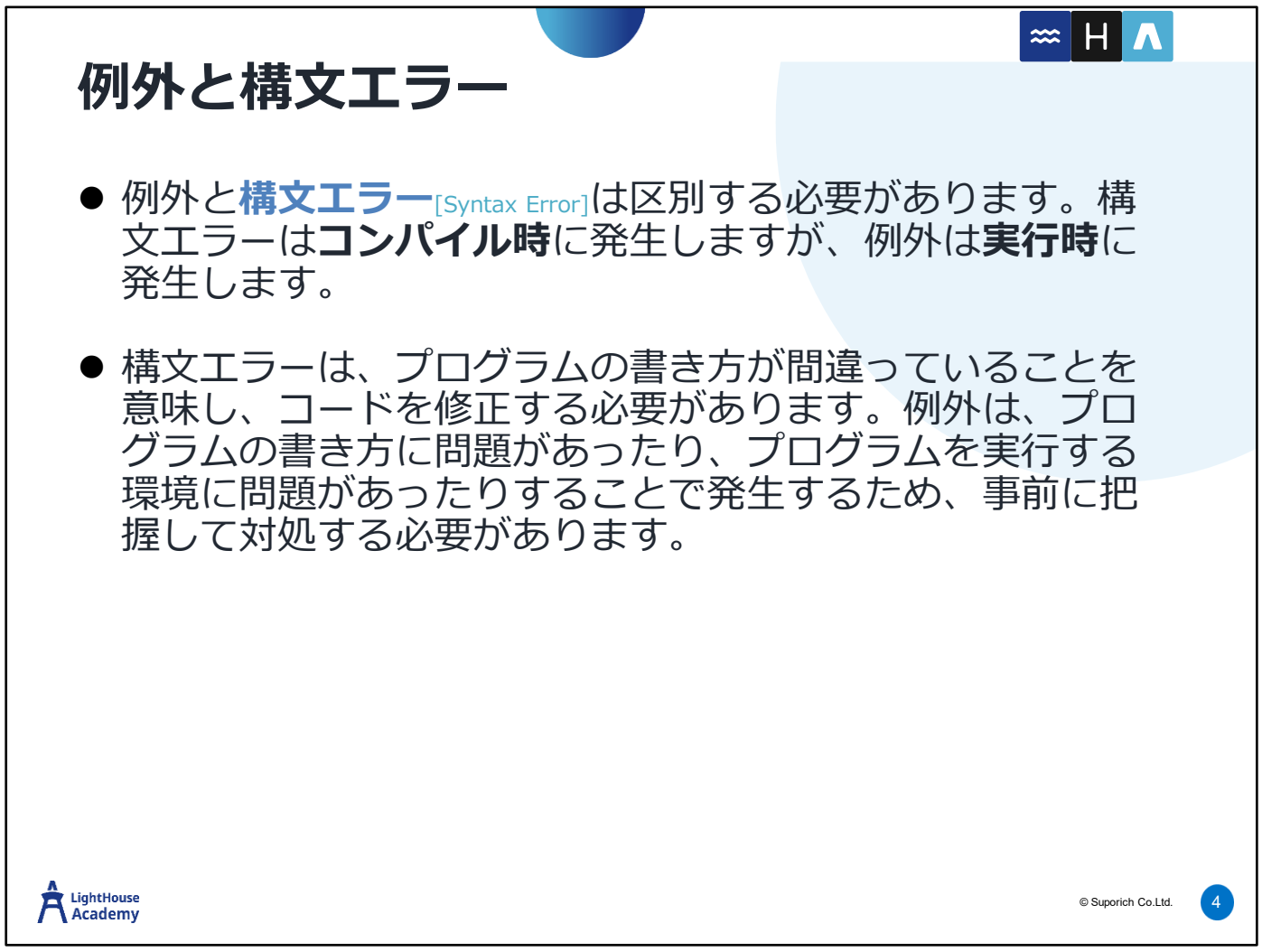
LightHouse Academy

例外

- **例外**[Exception]とは、プログラムの動作中に発生する、異常な状態や想定外の状態であり、特別な処理が必要な状態を指します。
- 一部の例外はプログラム内で処理する必要があります。 そうでない場合は、プログラムの実行を継続することができず、強制的に終了されます。

LightHouse Academy

プログラムを実行したときに発生するエラーのことを例外といいます。
例外が発生すると何が起こるのかというと、該当する、例外クラスのオブジェクトが自動生成される。



例外いわゆるプログラムを実行したときのエラーと構文エラー、って区別する必要があって、
構文エラーは、例えば、System.out.printlnのmがnになってたりとか、プログラムの書き方が間違っていること。
一方、例外処理は、プログラムの書き方に問題があったり、プログラムを実行する環境そのものに問題があったりすることで発生するエラーなので、事前に把握して対処する必要があります。

例外の発生と処理

- プログラムの早期終了につながる致命的なエラーを防ぐため、プログラミングでは、特殊な入力や環境問題が発生することを考慮し、例外を発生させる、いわゆる例外を**スロー** [Throw] する必要があります。

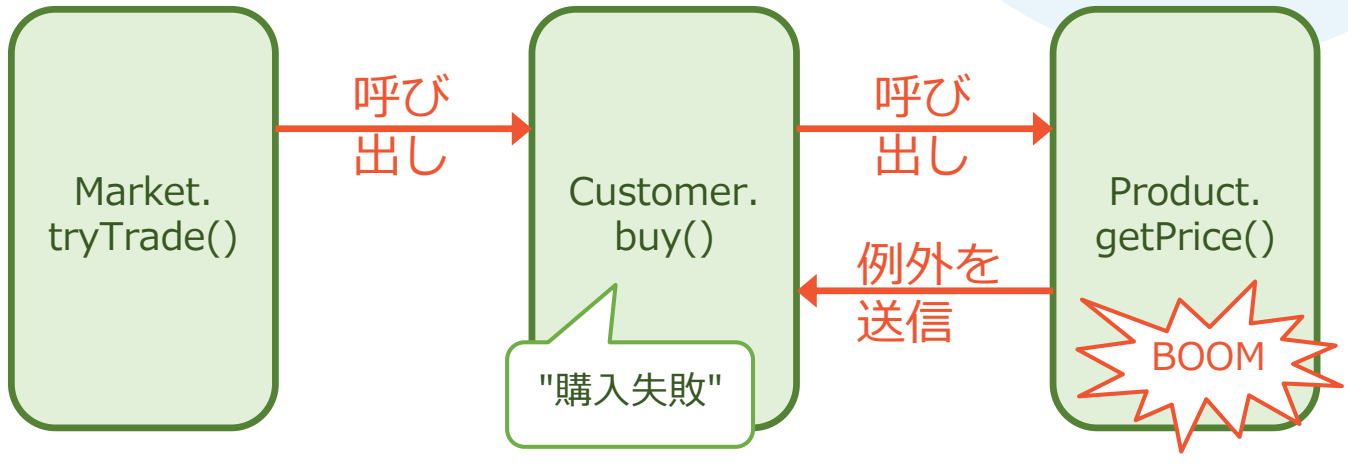
- 次に、例外が発生する可能性のある地点での処理方法には、一般的に 2 つの方法があります：
  - 例外の対処法がわかったら、**現在のメソッドに**直接コードを書いて問題を解決します。
  - 例外の対処法がわからないときは、現在のメソッドを**呼び出したメソッドに**例外を委ねます。

次へ

前へ

Example

取引システムにおいて、買い主が取引時に「商品の価格が設定されない」という問題が発生。




Q&A
LightHouse Academy
© Suporich Co.Ltd.
7

目次

LightHouse Academy

Java の例外

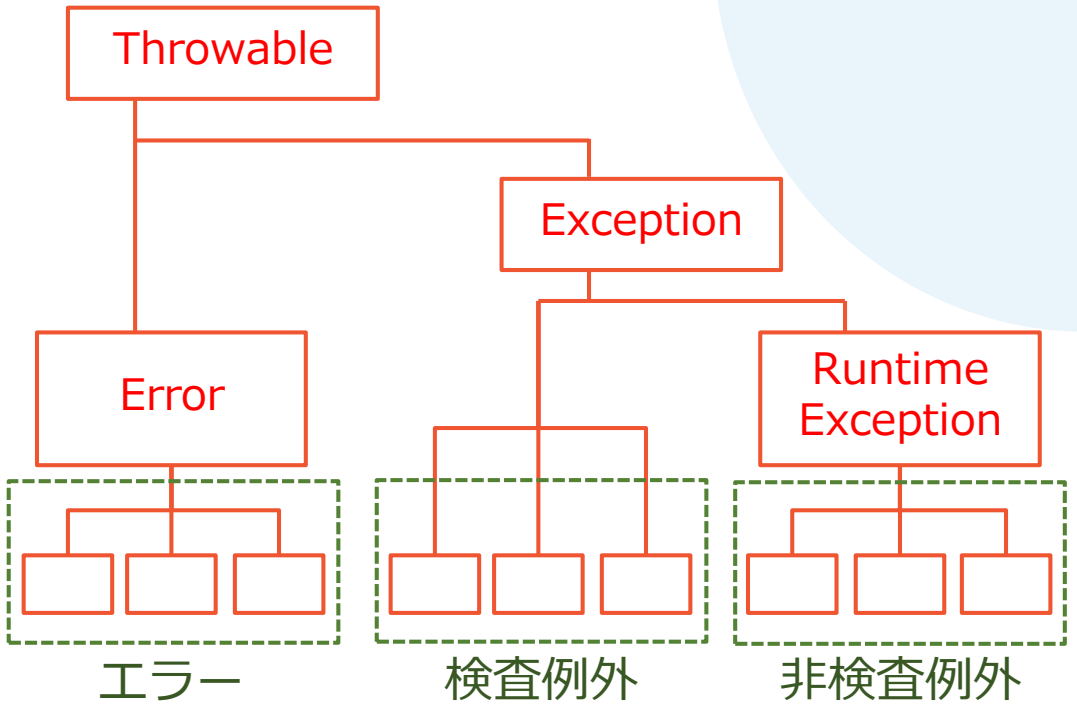
- Java におけるすべての例外は、一つの Java クラスで表現されます。 すべての例外クラスは **Throwable** クラスを継承しています。
- Java における例外は、大きく分けて「**エラー・Error**」、「**非検査例外・Runtime Exception**」、「**検査例外・Exception**」の 3 種類に分類されます。
- 次の図は、それらの関連性を示しています。

次へ

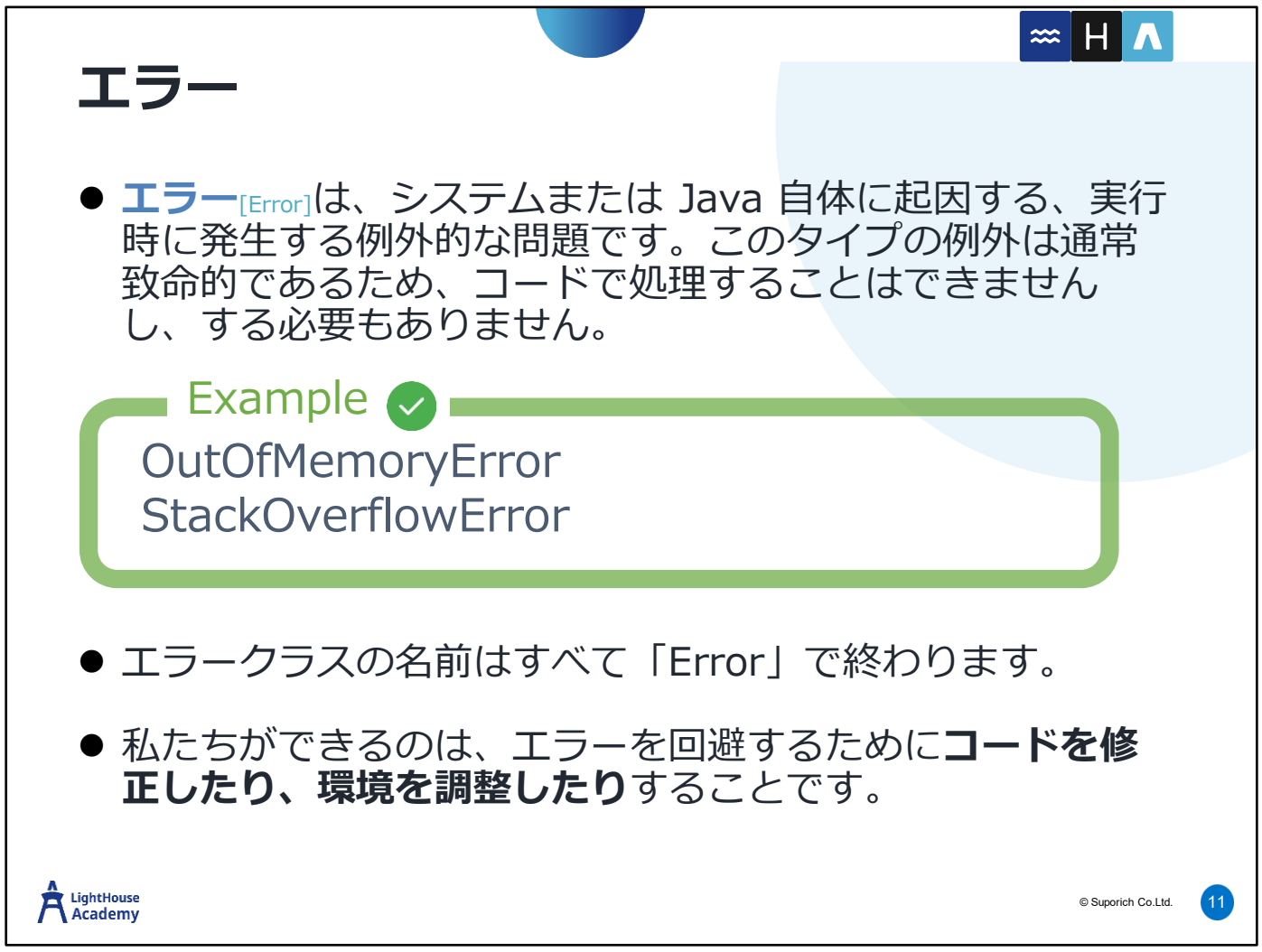
LightHouse Academy

例外が発生すると、例外クラスのオブジェクトが自動生成される
JAVAでは、発生したエラーの種類ごとに対応したクラスが分けられています。
なので、JAVAには、どういう例外クラスがあるのかわかると、どういう種類のエラーが発生しているのかがわかる
全ての例外のクラスの階層の一番上にあるのが、この、スローアブルクラスです。
全ての例外クラスは、この「スローアブルクラス」を継承して作られていると思ってください。
で、このスローアブルクラスを継承したものはたくさんあるですけど、その中で重要なクラスを３つだけ紹介します。

前へ



LightHouse Academy



例外処理は必須ではない
何故か：
JAVAのプログラムをいくらうまく書いていたとしても、メモリがなくなってしまうことは１００％防ぐことができない。だからここに関しては、例外処理をしなくても大丈夫

非検査例外

- **非検査例外**[Unchecked Exception]、または**実行時例外**[Runtime Exception]は、比較的基本的で頻繁に発生する例外です。
- 配列要素に対する操作、除算、メソッド呼び出しなど、多くの基本構文が非検査例外を発生する可能性があります。これらの例外をすべて処理すると、コードが余計に複雑になるので、非検査例外は**コードで処理する必要がない**です。
- 一方、非検査例外は、論理的な誤りやプログラム自体の考え方の甘さである可能性が高く、それを回避するために**コードを修正**することが必要です。
- 非検査例外クラスの名前はすべて「Exception」で終わり、そして **RuntimeException** クラスを継承します。

LightHouse Academy

ランタイムエクセプション
この、ランタイムエクセプションの対応も任意
例えば、算術例外なんか割り算とかを使うと、エラーが発生する可能性がある。
割り算をするために毎回、例外処理を書いてたら大変ですよね
なので、プログラムを書くときに気を付けましょうということで、任意になっています。

よくある非検査例外

- 次の表は、一般的な非検査例外の一覧です：

例外の種類	クラス名	例
算術例外	ArithmeticException	`double a = 1 / 0;`
配列のインデックスが範囲外の例外	ArrayIndexOutOf BoundsException	`int arr = new int[3];` `arr[3] = 1;`
ヌルポインターの例外	NullPointer Exception	`String a = null;` `int b = a.length();`
機能をサポートしない時の例外	Unsupported OperationException	`List list = List.of(1, 2);` `list.add(3);`
配列のサイズが負の例外	NegativeArraySize Exception	`Int[] a = new int [-2];`

検査例外

- **検査例外**[Checked Exception]とは、先ほど述べた 2 つの特殊なケース以外の一般的な例外を指します。ファイルの読み書き、ネットワーク通信、データベースへのアクセスなど、実際のあらゆる操作で発生する可能性があります。
- 検査例外が発生する可能性がある時点で処理しないと、エラーが発生し、プログラムが実行されません。つまり、必ず**コードで対処すべき**です。
- 検査例外クラスの名前はすべて「Exception」で終わっており、そして RuntimeException クラスを**継承していません**。
- 一般的な検査例外クラスには、FileIOException、FileNotFoundExeption などがあります。このほかにも、今後の授業で様々なものに出会います。

LightHouse Academy

・検査例外

　例えばファイルを入出力するようなプログラムを書く時には、きちんと例外処理を書かないと

　コンパイル時点でエラーになります。

まとめ：エラー・例外の種類

Sum Up

種類	スーパークラス	発生タイミング	処理方法
構文エラー	-	コンパイル時	コードの修正
エラー	Error	実行時	コードの修正または環境の調整
非検査例外	Runtime Exception	実行時	コードの修正または例外処理
検査例外	Exception	実行時	コードで例外処理

目次

LightHouse Academy

Java の例外処理

- Java では、Java 自身が持っているクラスやメソッドが何らかの例外を発生させることがあります。 また、後で使う外部パッケージも例外を発生させることがあります（通常は検査例外）。また、手動で例外を発生させることも可能です。

- この段階では、まず例外が発生しそうなときの対処法を覚えておく必要があります。

次へ

前へ

- Java では、例外処理に 2 つの選択肢があります：
  - **try-catch** 文は、現在のメソッド内で問題を解決するために使用されます。
  - **throws** キーワードは、現在のメソッドを呼び出したコードに例外を渡すために使用されます。

Note

検査例外は処理しなければならないが、非検査例外は処理してもよく、処理しなくてもいい。

try-catch 文

- 構文：

```
1 try {
2     codes;
3 } catch (Exception e) {
4     codes2;
5 }
```

- codes で、例外が**発生する可能性があるコード**を書いてください。codes2 で、例外の発生を**解決するコード**を書いてください。

LightHouse Academy

・例外処理の書き方
まず、はじめに例外が発生したらどうなるかというと
該当する例外のクラスのオブジェクトが自動生成されることをやりましたよね
で、この自動生成されるオブジェクトに対して何もしないと、ＪＡＶＡのプログラムは、異常終了するという動きになる。でも、プログラムは、勝手に止まったら困る
ということで、例外処理をかくのですけれども、例外処理時に自動生成される例外オブジェクトを受け取るために,try-catchブロックというものを書きます。
Tryの中には、通常の処理「本来プログラムで行わせたい処理」
この処理の中では、例外が発生するかもしれないんだけど、試しにやってみようという部分
試にやってみたんだけど、例外が発生しちゃった。処理がキャッチブロックに飛びます。
（例外クラス　オブジェクト名）オブジェクト名→例外オブジェクトを受け取るためのためのオブジェクト変数です。何を書くかというと、例えばエラーメッセージを画面に表示したりとか、エラーのログを取ったりとかです。

try-catch 文による例外処理

- この文に ArrayIndexOutOfBoundsException が発生：

```
1 int[] arr = new int[10];
2 System.out.println(arr[10]);
```

- エラーが発生する可能性のあるコードを try-catch 文に入れて処理：

```
1 int[] arr = new int[10];
2 try {
3     System.out.println(arr[10]);
4 } catch (Exception e) {
5     System.out.println("Failed to print number");
6 }
```

複数の catch ブロック

- コードで複数種の例外が発生する可能性がある場合、例外の種類に応じて異なるコードが必要になる場合があります。
- 例：

```
try {
    codes;
} catch (Exception1 e) {
    codes2;
} catch (Exception2 e) {
    codes3;
} catch (Exception3 e) {
    codes4;
}
```

- ここで、Exception1、Exception2、Exception3 は異なる例外のクラスです。

マルチキャッチ

- また、複数の例外を「|」で区切って記述すれば、複数の例外を同じ catch ブロックで処理することも可能である。 これを**マルチキャッチ**[Multiple Catch]と呼びます：

```
1 try {
2     codes;
3 } catch (Exception1 | Exception2 e) {
4     codes2;
5 } catch (Exception3 e) {
6     codes3;
7 } catch (Exception4 e) {
8     codes4;
9 }
```

try-catch 文の実行順序

- try-catch 文の実行順序に注意してください。try ブロックの実行中に例外が発生した場合、try ブロックは**直ちに終了**します。その後、例外の型に合致する catch ブロック内の文のみが実行されます。try ブロックに残されたコードは、実行されません。

- try ブロックが例外なく実行された場合、コードは正常に終了し、catch ブロックは実行されません。

finally ブロック

- 例外が発生してもしなくても、ある文を実行したい場合があります。
- 例えば、try ブロックの中で、いくつかのファイルリソースをメモリに入れることがあります。 これらのリソースは、**例外が発生するかどうかにかかわらず、コードの終了時に解放**されるようにしたい。
- この場合、**finally** ブロックを使用することができます：

```
try {
    codes;
} catch (Exception e) {
    codes2;
} finally {
    codes3;
}
```

LightHouse Academy

・ファイナリー
例外が発生してもしなくても行わせたい後処理を書く。

finally ブロックの実行順序

- どのような場合でも、finally ブロック内の文は try-catch 文が終了した後に必ず実行されます。
- try ブロックや catch ブロック内で **return 文ですぐにメソッドを終了**させても、finally ブロック内のコードは（戻り値が使われる前に）実行されます。

- また、確実にリソースを解放する方法として、try-with-resource 文があります。これについては、後ほど実際に使用する際に説明します（§ 3.3.3）。

throws キーワード

- 現在のメソッドで、例外を解決する方法がわからない場合、**throws** キーワードを使って例外を**呼び出し側に渡し**、処理させることができます。
- 例外をスローするには、メソッド定義の括弧「)」の後に throws キーワードと例外のクラス名を追加します：

```
1 public static void test() throws ArrayIndexOutOfBoundException {
2     int[] arr = new int[10];
3     arr[10] = 100;
4 }
```

- また、カンマで区切って、複数の例外名を記述することもできます：

```
void test() throws IOException, ArithmeticException {}
```

LightHouse Academy

P30のスライドを使用して説明する

throws キーワードの仕組み

- throws キーワードを持つメソッドで例外が発生した場合、メソッド自体が直ぐに終了してしまいます。同時に、そのメソッドが呼び出された場所でその例外が発生されます。

- したがって、throws キーワードを持つメソッドを使うとき、**それ自体に例外が発生**するようになっています。呼び出し側は、同じ方法（try-catch か throws か）で例外処理する必要があります。

throw 文

- **手動で例外をスロー**（発生）する場合は、**throw** 文を使用することができます。

```
throw new Exception();
```

- Exception は他の例外クラスで置換することができます。また、Exception を継承したオリジナルの例外クラスを定義して使用することも可能です。

- throws 文との違いに注意しましょう。

Throwとthrowsの違いと補足

- throw　例外オブジェクト：例外を強制的に発生させる
- throws 例外クラス :メソッドが例外を発生させることを宣言する

throwの例

```
try {
        throw new ArrayIndexOutOfBoundsException();
    }catch(ArrayIndexOutOfBoundsException e) {
        System.out.println("添え字に問題がありそうです。");
    }
```

throwsの例

代わりにメソッド定義のところに記載することで、try-catchを書かなくてもコンパイルが通るようになる

```
public static void arrayEx() throws ArrayIndexOutOfBoundsException {
        throw new ArrayIndexOutOfBoundsException();
}
```

例外を強制的に発生させている
Try-catchが必要

まとめ

Sum Up

1.例外の定義と例外処理の考え方：その場で解決するか、呼び出し側に任せるか。

2.Java における例外の種類：
- ① エラー：処理できない。
- ② 非検査例外：処理しなくてもよい。

3.Java における例外処理の方法：
- ① その場で解決する：try-catch-finally 文。
- ② 呼び出し側に任せる：throws キーワード。

Light in Your Career.

THANK YOU!