

# Extensible Exception

kbkz.tech #10

吉村 優

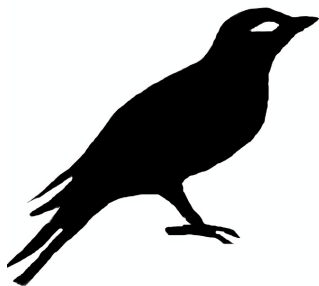
[https://twitter.com/\\_yyu\\_](https://twitter.com/_yyu_)

<http://qiita.com/yyu>

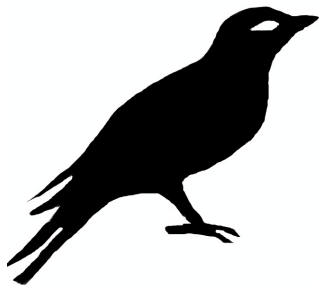
<https://github.com/y-yu>

July 16, 2016

# 自己紹介

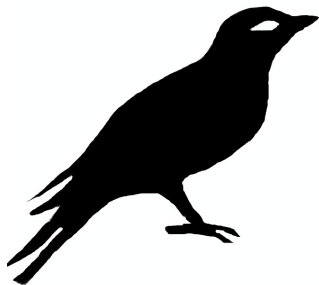


# 自己紹介



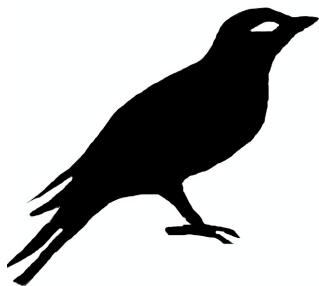
- 筑波大学 情報科学類 学士  
(COINS11)

# 自己紹介



- 筑波大学 情報科学類 学士 (COINS11)
- 現在は Scala を書く仕事に従事

# 自己紹介



- 筑波大学 情報科学類 学士 (COINS11)
- 現在は Scala を書く仕事に従事
- エラー処理に関する話をします

# エラー値とは？

# エラー値とは？

エラー値

# エラー値とは？

## エラー値

- エラーであることを表す値



# エラー値とは？

## エラー値

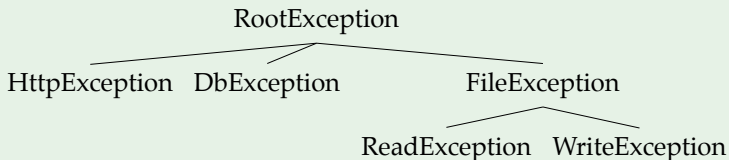
- エラーであることを表す値
- 階層構造（木構造）になるのが一般的

# エラー値とは？

## エラー値

- エラーであることを表す値
- 階層構造（木構造）になるのが一般的

## エラー値の階層構造の例

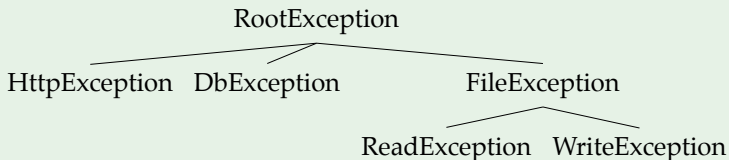


# エラー値とは？

## エラー値

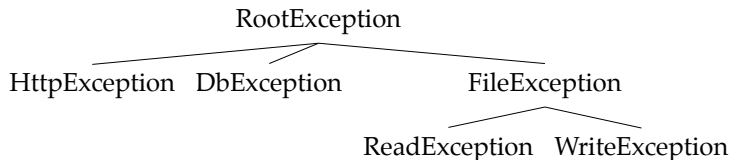
- エラーであることを表す値
- 階層構造（木構造）になるのが一般的

## エラー値の階層構造の例

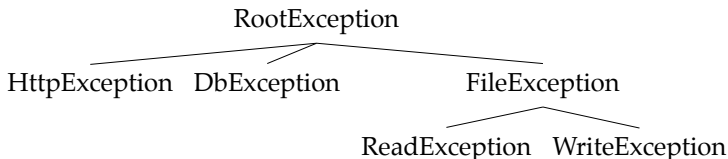


どうやって階層構造を作る？

# 継承を用いた表現

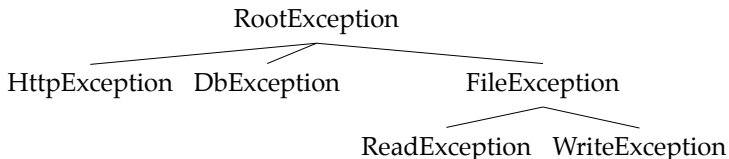


# 継承を用いた表現



```
trait RootException extends Throwable
case class DatabaseException(m: String) extends RootException
case class HttpException(m: String) extends RootException
trait FileException extends RootException
case class ReadException(m: String) extends FileException
case class WriteException(m: String) extends FileException
```

# 継承を用いた表現



```
trait RootException extends Throwable
case class DatabaseException(m: String) extends RootException
case class HttpException(m: String) extends RootException
trait FileException extends RootException
case class ReadException(m: String) extends FileException
case class WriteException(m: String) extends FileException
```

どうして継承を使うの？

# サブタイプ多相

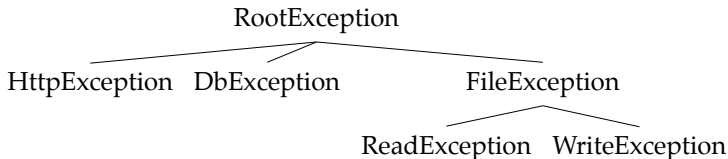
“ 型  $A$  が型  $B$  の *subtype* (部分型) のとき、型  $B$  の式を書くべきところに、型  $A$  の式を書いても良い。 ”

筑波大学 プログラム言語論 [1]

# サブタイプ多相

“ 型  $A$  が型  $B$  の *subtype* (部分型) のとき、型  $B$  の式を書くべきところに、型  $A$  の式を書いても良い。 ”

筑波大学 プログラム言語論 [1]

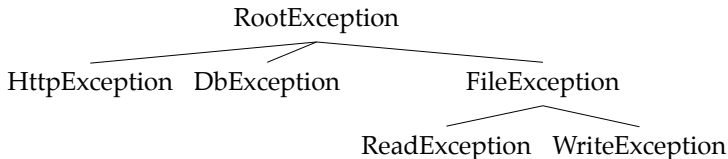




# サブタイプ多相

“ 型  $A$  が型  $B$  の *subtype* (部分型) のとき、型  $B$  の式を書くべきところに、型  $A$  の式を書いても良い。 ”

筑波大学 プログラム言語論 [1]

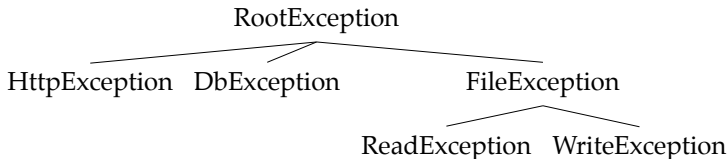


例

# サブタイプ多相

“ 型  $A$  が型  $B$  の *subtype* (部分型) のとき、型  $B$  の式を書くべきところに、型  $A$  の式を書いても良い。 ”

筑波大学 プログラム言語論 [1]



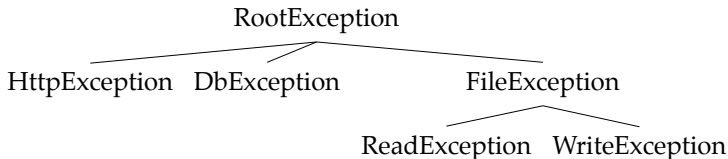
## 例

- `RootException` を書くべきところに `HttpException` を書く

# サブタイプ多相

“ 型  $A$  が型  $B$  の *subtype* (部分型) のとき、型  $B$  の式を書くべきところに、型  $A$  の式を書いても良い。 ”

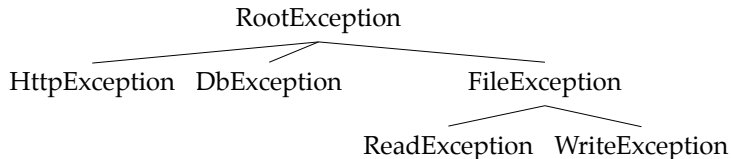
筑波大学 プログラム言語論 [1]



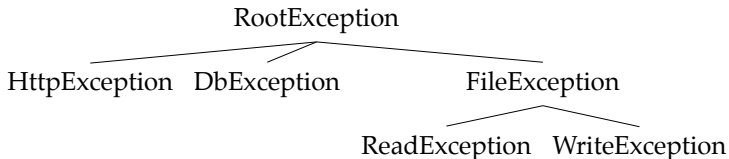
## 例

- `RootException` を書くべきところに `HttpException` を書く
- `RootException` を書くべきところに `ReadException` を書く

# 階層構造の変更

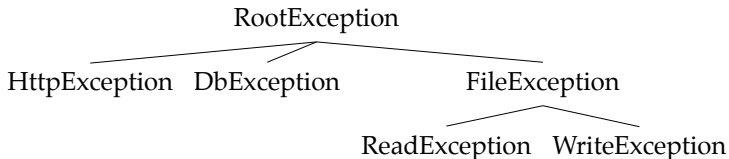


# 階層構造の変更

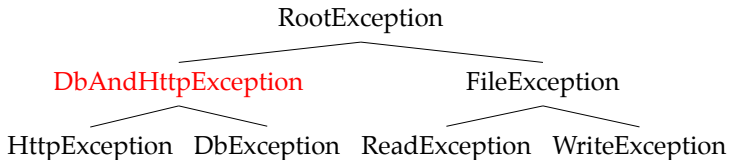


後からこの階層構造を変更できる？

# 階層構造の変更



後からこの階層構造を変更できる？



無理では？

無理では？

継承でやるのはよくない？



無理では？

継承でやるのはよくない？

型クラスでやろう！



# 新しい型クラス

# 新しい型クラス

## 変換を表す型クラス:~>

型  $A$  から型  $B$  への変換ができることを表す型クラス

# 新しい型クラス

## 変換を表す型クラス:~>

型  $A$  から型  $B$  への変換ができることを表す型クラス

```
trait ~>[-A, +B] {  
  def apply(a: A): B  
}
```

# 新しい型クラス

## 変換を表す型クラス:~>

型  $A$  から型  $B$  への変換ができることを表す型クラス

```
trait :~>[-A, +B] {  
  def apply(a: A): B  
}
```

:~>のインスタンスとして、階層構造を定義

# 新しい型クラス

## 変換を表す型クラス:~>

型  $A$  から型  $B$  への変換ができることを表す型クラス

```
trait :~>[-A, +B] {  
  def apply(a: A): B  
}
```

:~>のインスタンスとして、階層構造を定義

## 例

```
implicit val db = new (DbException :~> RootException) {  
  def apply(a: DbException): RootException =  
    new RootException { ??? }  
}
```

# 階層構造の拡張

# 階層構造の拡張

- 1 新しいエラー値を定義



# 階層構造の拡張

- ① 新しいエラー値を定義
- ② 型クラス:~>のインスタンスを定義

# 階層構造の拡張

- ❶ 新しいエラー値を定義
- ❷ 型クラス:~>のインスタンスを定義
- ❸ 型クラス:~>のインスタンスを使うように Either を改造

# 新しいエラー値の定義

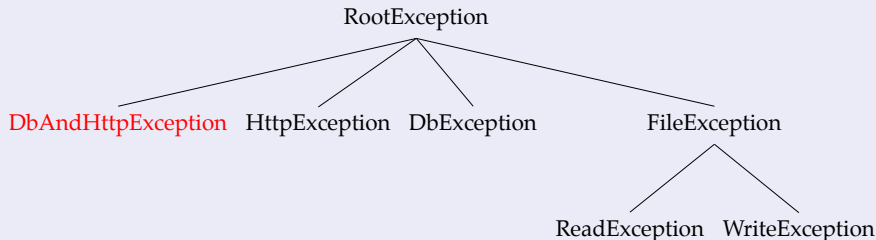
# 新しいエラー値の定義

```
case class DbAndHttpException(m: String) extends RootException
```

# 新しいエラー値の定義


```
case class DbAndHttpException(m: String) extends RootException
```

## 継承に基づく階層構造



# 参考文献

 亀山幸義.  
プログラム言語論 オブジェクト指向, 2015.

 Simon Marlow.  
An extensible dynamically-typed hierarchy of exceptions.  
*In Proceedings of the 2006 ACM SIGPLAN Workshop on Haskell,*  
*Haskell '06*, pp. 96–106, New York, NY, USA, 2006. ACM.

Thank you for listening!  
Any question?