独習JAVA第3版

8.4 例外とエラークラス
8.5 throws ステートメント
8.6 独自の例外

Throwableコンストラクタ

- catchブロックにはThrowable型のパラメータが必ず1つなければならない
- Throwableコンストラクタ
  - Throwable()
  - Throwable(String *message*)

messageには問題を通知する文字列のメッセージ

Throwableクラスのメソッド

- Throwableクラスのメソッドは次の2つ
  - getMessage()メソッド
    構文：String getMessage()
    説明：コンストラクタから提供された文字列を返す
  - printStackTrace()メソッド
    構文：void printStackTrace()
    説明：問題が発生した時点でのスタック情報を表示

エラークラス(1/3)

- Errorクラス
  - Throwableクラスを拡張する
    JVMで検出される可能性がある重大な問題を表す
    サブクラスが10以上ある

エラークラス(2/3)

- Exceptionクラス
  - 構文： Exception()
    Exception(String *message*)
  - 説明：
    Throwableクラスを拡張する
    実行時に発生する可能性がある様々な問題を表す
    サブクラスがある

messageは問題を通知する
文字列のメッセージ

エラークラス(3/3)

- RuntimeExceptionクラス
  - 構文: RuntimeException()
    RuntimeException(String *message*)

messageは問題を通知する
文字列のメッセージ

  - 説明:
    Exceptionの最も重要なサブクラス
    プログラムの実行中に発生する頻度の高い問題を表す

throwsステートメント(1/3)

- 呼び出し元に対して例外を投げる
  可能性があるメソッドを書いた場合、
  どの例外が投げられるのか示しておくと便利

  → 投げられる例外を明示するには
  コンストラクタ内のthrowsステートメントを使用

throwsステートメント(2/3)

- コンストラクタ内のthrowsステートメント

```
consModifiers clsName(cparams) throws exceptions {
    // コンストラクタの本体
}
```

consModifiers：修飾子のリスト（省略可能）

clsName　：クラス名

cparams　：コンストラクタのパラメータ（省略可能）

exceptions　：このコンストラクタから投げられる可能性がある
　　例外クラスをカンマで区切って並べたもの

throwsステートメント(3/3)

- メソッドの宣言でのthrowsステートメントの記述

```
mthModifiers rtype mthName(mparams) throws exceptions {
  // メソッドの本体
}
```

mthModifiers：修飾子のリスト（省略可能）
rtype：mthNameというメソッドの戻り値の型
mparams：パラメータ（省略可能）
excptions：このメソッドで投げられる可能性がある
　　例外クラスをカンマで区切って並べたもの

throwsステートメントの例

```
class ThrowsDemo {
  public static void main(String args[]) {
    a();
  }

  public static void a() {
    try {
      b();
    }
    catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
```

例外を捕獲

```
public static void b() throws
  ClassNotFoundException {
    c();
  }

public static void c() throws
  ClassNotFoundException {
    Class cls =
        Class.forName("java.lang.Integer");
    System.out.println(cls.getName());
    System.out.println(cls.isInterface());
  }
}
```

throwsステートメントを宣言

例の説明

- 例ではmain()メソッドからa()メソッドを呼び出し、a()メソッドからb()メソッド、b()メソッドからc()メソッドを呼び出している
- ClassNotfoundException例外が投げられる可能性がある
- この例外はRuntimeExcptionのサブクラスではないため、投げられる可能性がある全てのメソッドで例外を捕獲または宣言する必要がある

独自の例外

- Javaには、Exceptionのサブクラスとして独自の例外クラスを作成する機能がある
- 例外のインスタンスを投げるにはthrowsステートメントを使用

独自の例外の例

```
import java.util.*;

class ExceptionSubclass {
  public static void main(String args[]) {
a();
  }
  static void a() {
    try {
      b();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  static void b() throws ExceptionA {
    try {
      c();
    }
     catch (ExceptionB e) {
      e.printStackTrace();
    }
  }
  static void c() throws ExceptionA, ExceptionB {
    Random random = new Random();
    int i = random.nextInt();
    if (i % 2 == 0) {
      throw new ExceptionA("We have a problem");
    }
    else {
      throw new ExceptionB("We have a big problem");
    }
  }
}

class ExceptionA extends Exception {
  public ExceptionA(String message) {
    super(message);
  }
}

class ExceptionB extends Exception {
  public ExceptionB(String message) {
    super(message);
  }
}
```

例の説明

- ExceptionAクラス、ExceptionBクラスは
  Exceptionクラスを拡張
- c()メソッドはランダムでどちらの例外を
  投げるか決める
- b()メソッドではthrowsステートメントを使って
  ExceptionA例外を宣言、
  ExceptionB例外はcatchブロックで捕獲
- a()メソッドでは全ての例外を捕獲

課題

```
class ErrorTest {
  public static void main( String args[] ) {
    int m1, m2 ;
    try {
      m1 = Integer.valueOf(args[0]).intValue() ;
      m2 = Integer.valueOf(args[1]).intValue() ;
    }
    catch( ArrayIndexOutOfBoundsException e ) {
      e.printStackTrace() ;
    }
    try{
      a( m1, m2 ) ;
    }
    catch( ArithmeticException e ) {
      e.printStackTrace() ;
    }
  }
```

これは二つの整数を受け取り、
それをもとに除算するプログラムである。
引数が足りないとき、0で除算することになったとき
例外処理を行っているが、問題点がある。
その問題点を指摘し、改善せよ。

```
  public static void a( int a1, int a2 ) {
    try {
      b( a1, a2 ) ;
    }
    catch( NullPointerException e ) {
      e.printStackTrace() ;
    }
  }

  public static void b( int b1, int b2 ) {
    int i = b1 ;
    int j = b2 ;
    try {
      System.out.println( (float)i/j );
    }
    catch( NullPointerException e ) {
      e.printStackTrace();
    }
  }
}
```