Introduction to Java

Special Care Set for Java Begineers/ Ex 12

・Notice!
The motivation of this page is to lead regular exercises.
We prepare many useful information in this page as hints of exercises, but this page has no answer.
If you have question, please don't hesitate to ask and discuss your teacher and teaching assistants.
このページは、通常の演習問題への導きとして作られています。
多くの有用な情報をヒントとして準備しておりますが、ここに回答が隠されているとかはありません。
もし、演習に対して質問等ございましたら、遠慮なく教員・TAに質問・議論をしてください。

Example Problem: 例外処理を定義する
Define User Exception

先週は、既存の例外処理と、例外処理の受け取り方（Try-Throw-Catch-Finally）について勉強しました。
例外処理は、既知・未知のリスクに関わらず、『例外』と捉えられる微細なバグに対する処理を記述し、その時の対処を行うことが出来るようにしています。
それによって、C言語等のレガシーなエラー処理で煩雑になっていたコードをまとめることが出来たり、その時点その時点で対応に漏れることなく、ソフトウエアを不用意に停止することなく、動かし続けることが可能となっています。

では、自分で例外処理を記述する場合はどうすればいいでしょうか？
前回までは、例外に当たる内容は全てJava APIに既に登録されている『出来合いの』例外でしたが、ソフトウエアを構成している時には当然ながら『出来合い』ではない例外が非常に多く出てくることと思います。
今回は、そのような例外を定義するクラスをどのように作るかを学んでいきます。

1.『例外処理』のためのクラスの作成
Creating class object for "Exception"

例外やエラー処理を行う場合、右図のような階層構造を持つ『Throwable（異常としてスロー可能）』なクラス構造を持ちます。
一般クラスの最上位がObjectクラスである（何もクラスの継承を行わない場合、Objectクラスを継承する）のと同様に、例外やエラー処理を行う場合の最上位はThrowableクラスが最上位であり、『例外処理』のためのクラスと言う意味では『Exception』クラスが最上位となります。
特にIOExceptionやRuntimeExceptionなど、特別な意味を持つ既存の例外と役割が同一ではない限り、新しいExceptionを定義するクラスは『Exception』クラスを継承して作成します。

以下は、MyExceptionとして、『とりあえず何が悪い訳ではないけれども新しい例外として処理

する』クラスを作成しています。

```
class MyException extends Exception{
        MyException(Throwable cause){
                super(cause);
        }
        MyException(){
                super();
        }
        MyException(String error){
                super(error);
        }
        MyException(String error, Throwable cause){
                super(error,cause);
        }
}
```

上位クラスであるjava.lang.Exceptionは、次の4つのコンストラクタを持つことがJava APIからわかります。

コンストラクタの概要
Exception() 詳細メッセージに null を使用して、新規例外を構築します。
Exception(String message) 指定された詳細メッセージを使用して、新規例外を構築します。
Exception(String message, Throwable cause) 指定された詳細メッセージおよび原因を使用して新規例外を構築します。
Exception(Throwable cause) 指定された原因と詳細メッセージ (cause==null ? null : cause.toString()) を持つ、新しい例外を構築します (通常、クラスと原因の詳細メッセージを含みます)。

全てを定義して、上位クラスに預ける必要はありませんが、例のように、コンストラクタを生成した後、上位クラスへとsuper()を用いてコンストラクタの委譲を行うことが出来ます。

2. 状態を記述できる『MyException』の構築
Construct "MyException" which is able to describe state

さて、先ほどのMyExceptionでは、ただExceptionと全く同じような動作をする別の例外クラスを作成した状態になっています。
そこで、MyExceptionでは、『何のエラーが起きたのか』を、エラーコードで保持して、後で知ることが出来るような例外処理にしてみたいと思います。
エラーコードにあたる部分を新たにメンバー変数として定義し、getCode()メソッドを作ることにしましょう。

```
class MyException extends Exception{
        private int code;
        public MyException(int code, String error){
                super(error);
                this.code = code;
        }
        public int getCode(){
                return code;
        }
}
```

このMyExceptionを使用するようなサンプルコードを次のように示します。
仕様は、"『3』の倍数か『5』の倍数にはアホになる=Exception"を吐くようにします。

```
public class ExceptionTest {

        public static void main(String[] args) {

                for(int i = 1; i <= 20; i++){
                        try{
                                if(i % 15 == 0) throw new MyException(i, "You Fool!");
                                else if(i % 3 == 0) throw new MyException(i, "Fool!");
                                else if(i % 5 == 0) throw new MyException(i, "Fool!");
                                else System.out.println(i);
                        }catch(MyException me){
                                System.err.println(me.getCode() + " " + me.getMessage());
                        }

                }
        }
}

class MyException extends Exception{
        private int code;
        public MyException(int code, String error){
                super(error);
                this.code = code;
        }
        public int getCode(){
                return code;
        }
}
```

上のソースコードの出力結果は次の通りです。ただし、System.out.printlnとSystem.err.printlnの出力は同期しませんので、実行の度に出力のタイミングが変化します。

```
1
2
3 Fool!
5 Fool!
6 Fool!
9 Fool!
10 Fool!
12 Fool!4
7
8
11
```

13
14
15 You Fool!
16
17
18 Fool!19

20 Fool!