例外処理（Exception）

-- try chatch--

Java の例外処理

- プログラミングを行う際には実行時に対するエラー（例えば、ファイルが存在しないとか、配列のアドレス先間違いなど）を適切に処理する仕組みを実装する必要がある。
- ｃ言語など他の多く手続き型言語では関数の戻り値などを確認することでエラーの検出を行っている。この場合エラーに対する適切な処理を行うためには if 文で制御しなければならない。これはプログラムの複雑化を招く要因になる。
- Java では例外の発生に際して throw と catch を使ったエラー処理プロトコルがあらかじめ用意されている。
  - メソッド内で何らかのエラーが発生するとそのエラーが throw されるという。
  - 反対にそのエラーを検出するメソッドを呼び出すことをエラーを catch するという。

Exception クラス

- Java は Exception クラスを Throwable クラスから派生している（API ドキュメントクラス階層参照）。
- Throwable クラスは 2 つのサブクラス Error と Exception を直下に持っている。
- Error クラスは Java のランタイムが修復不可能なエラーをインタプリタに対して throw するために使用するためプログラムで処理することは不可能。
- Exception クラスのサブクラスについては RuntimeException クラスなのかそれ以外のサブクラスなのかでプログラム内で対応が必要かどうかが変わる。
  - RuntimeException クラス：例外処理は任意
  - それ以外のサブクラス：例が処理は必須（コンパイラが許さない）

例外の検出・受信・対処（発生した例外に対処するための例外処理を書く）

- 例外発生の検出、受信および対処方法は以下のように **try - catch** ブロックで行われる。

```
try {
    例外が発生しそうな処理
} catch (例外クラス型１　引数) {
    例外処理１
} catch (例外クラス型２　引数) {
    例外処理２
} finally {
    共通処理（省略化）
}
```

- 発生する例外を全てキャッチしたい場合は、全ての例外クラスのスーパークラスである Exception 型で catch することができる。ただし、Exception 型は最後に記述する必要がある。Exception 型を最初に記述してしまうと、全ての例外オブジェクトがそこでキャッチされてしまうからである。なお、Exception 型の記述はあまり推奨される方法ではない。catch ブロックに記述する例外クラス型はなるべく限定できるものを記述すべきである。

参考 URL：Java の道　http://www2s.biglobe.ne.jp/~yuuki_ki/java_exception1.htm

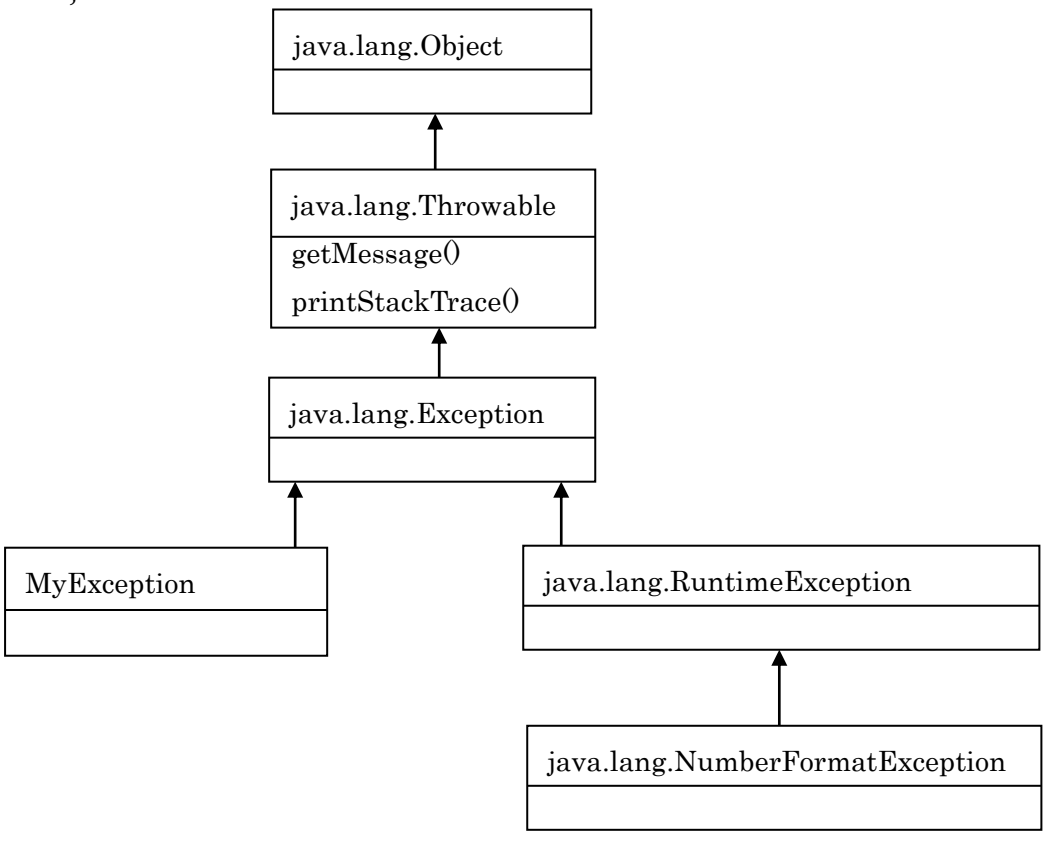
例外の発生（例外を意図的に発生させる方法）

- 例外を発生させるには throw キーワードを用いる
- throw の使い方
  - throw new 例外クラス名();
- 例外クラス名は Java 標準の例外クラスと共に独自に定義したクラスを使用することもできる

throw を使ったプログラムの例

起動時に引数が指定されなかったときに例外を発生しメッセージを表示する

```
//Throw1.java
public class Throw1 {
        public static void main(String[] args) {

                try {

                        // 引数が無いときには例外を発生する
                        if (args.length<1)
                                throw new Exception("引数がありません。");

                } catch (Exception e) {

                        // 例外のメッセージを表示する
                        System.out.println(
                                "メッセージ「"+e.getMessage()+"」 ");

                        // 例外の発生箇所や内容を表示する
                        e.printStackTrace();

                }
        }
}
```



例外を投げる

前の例では main メソッドで発生した例外を main メソッドの中で処理していましたが、発生した例外をメソッドの外(呼び出し元)で処理したいこともあります。それには thorows というキーワードを用いて例外を外に投げます。

例外をメソッドの外に出す例（コンストラクタで発生した例外を main メソッドで処理）

```
public class Throw3 {

	// method メソッドを呼び出し，例外処理を行う
	public static void main(String[] args) {
		try {

			// 例外を発生するコンストラクタの呼び出し
			Throw3 tmp=new Throw3(args);

		} catch (Exception e) {
			System.out.println(
				"メッセージ「"+e.getMessage()+"」");
			e.printStackTrace();
		}
	}

	// 例外を発生するコンストラクタ
	Throw3(String[] args) throws Exception {

		// 引数が無いときには例外を発生する
		if (args.length<1)
			throw new Exception("引数がありません。");
	}
}
```

メソッドと違ってコンストラクタには戻り値がないためコンストラクタの処理中に何か問題が起きても戻り値を使って呼び出し元に通知することができません。このような時コンストラクタでは戻り値の代わりに例外を使って問題を通知します。

独自の例外を定義 (以下の実習にはコードの誤りが何点かありますので適切に修正してください)

Java 標準の例外クラス（Exception クラス）を継承した独自の例外クラスを定義することができる。

```
// 独自の例外クラス例１：
// Exception クラスのサブクラスにする
class MyException extends Exception {
	public MyException() {
		super("独自の例外");
	}
}

// 独自の例外クラスを使うプログラム
public class Throw4 {

	public static void main(String[] args) {
		try {

			// 引数が無いときには例外を発生する
			if (args.length<1)
				throw new MyException();

		} catch (Exception e) {
			System.out.println(
				"メッセージ「"+e.getMessage()+"」 ");
			e.printStackTrace();
		}
	}
}

// 独自の例外クラス例２：
// ファイルコピーを行うクラスの例外処理

import java.io.*;

class FileNotOpenedException extends Exception {
	public FileNotOpenedException() {
		super("独自例外：ファイルが開けません");
	}
}

// 独自の例外クラスを使うプログラム
public class FileCopy {

	public static void main(String[] args) {
		//ファイル名の取得
		String inFile=args[0];
		String outFile=args[1];
		//ファイルのコピー
		try {
			fileCopy(inFile,outFile);
		} catch (FileNotFoundException e) {
			System.out.println("ファイルが見つかりません");
		}
		} catch (FileNotOpenedException e) {
			System.out.println("ファイルがオープンできません");
		}
		} catch (IOException e) {
			System.out.println("アクセス中にエラーが発生しました");
		}
	}
```

```
        static void fileCopy(String inFile,String outFile)
            thorows FileNotFOundException,FilrNotOpenedException,IOException {

                FileInputStream fis;
                FileOutputStream fos;

                //ファイルストリームのオープン
                fis = new FileInputStream(inFile);
                try {
                    fos = new FileOutputStream(outFile);
                } catch (FileNotFoundException e) {
                    throw new FileNotOpenedException();
                }
                //ファイルのコピー
                int c;
                while((c = fis.read()) != -1) {
                    fos.write(c);
                }
                //ファイルストリームのクローズ
                fis.close();
                fos.close();
        }
}
```

上記の例では入力ファイルが見つからなかった場合と、出力ファイルがオープンできなかった場合に発生する例外である FileNotFoundException を区別するために、出力ファイルがオープンできなかった場合に対処する FileNotOpendException を定義している。

fos = new FileOutputStream(outFile);この命令の FileNotFoundException を一度キャッチし新たに作成した例外を投げている。

以上のように例外が発生しそうな処理では、以下のどちらかの対処法が必要になります。

（１） try 節で例外が発生する処理を囲み catch 節で例外の対処法を記述する。

（２） メソッドの呼び出し元へ throws 節で例外を投げ返す。このとき独自の例外を定義することも可能である（throw new 新たな例外クラス)。

（３） throws 節で例外を投げ返すクラスを実体化しようとするクラスのコンパイル時は例外ハンドラ（try～catch 節）の記述が強制される。

Exception 実習

引数として main メソッドに与える Private IP アドレスが適切であるかチェックするための PrivateIpCheck クラスを実装する

○PrivateIpCheck クラスの仕様

package tokyopc.pinfo.util.connection とすること

PrivateIpFormatException を throw する

localhost あるいは xxx.xxx.xxx.xxx の形でないときこの例外を投げる

PrivateIpZoneException を throw する

Private アドレスの範囲内にないときこの例外を投げる

○PrivateIpFormatException クラスの仕様

Exception クラスへ適切なメッセージを投げ出力すること

○PrivateIpZoneException クラスの仕様

Exception クラスへ適切なメッセージを投げ出力すること

○QuerySwingViewControl クラス（呼び出し側クラス）の変更点

main メソッドの引数で取得した IP アドレスを PrivateIpCheck クラスを用いて評価し不適切であれば stackTrace を出力して終了すること

出力例

注意：a.b.c.d の場合はエラーメッセージだけ出力して処理は継続させること