http://d.hatena.ne.jp/fumokmm/20110420/1303271968
http://itpro.nikkeibp.co.jp/article/COLUMN/20120417/391316/

String で switch 文

マルチキャッチ

型推論

```
Map<String, List<String>> map = new HashMap<>();
```

右辺は書かなくてもわかるでしょ？　ってこと。

2 進数リテラルと数値のアンダースコア区切り

```
1234_5678
1_2_3_4__5_6_7_8L // アンダースコアはいくつあっても OK のようです
0b0001_0010_0100_1000
3.141_592_653_589_793d
0x1.ffff_ffff_ffff_fP1_023 // Double.MAX_VALUE
```

数値をわかりやすいようにアンダースコアを使って好きな場所で区切っても良いってこと

try with resources

try 句 で利用するリソースを自動的に close する機能。
いちいち finally 句で close する手間が省ける。
ただし、

1. try
2. close
3. catch
4. finally

の順で処理されるので、catch 後に finally でロールバックする等の処理をする場合は注意が必要。

動作順検証 例外が発生しない場合

出力結果

```
Constructor 11541827
Constructor 5324016
Constructor 24622029
try in
close 24622029
close 5324016
close 11541827
finally
```

動作順検証 try 句または close 処理中に例外が発生する場合

出力結果

```
Constructor 29194312
Constructor 2352593
Constructor 12910198
try in
close 12910198
close 2352593
close 29194312
catch
 java.lang.Exception: Exception in close 2352593
finally
```

close の途中で例外が発生しても、全ての close が呼ばれる

動作順検証 リソースのコンストラクタで例外が発生した場合

出力結果

```
Constructor 24622029
Constructor 29194312
close 24622029
catch
 java.lang.Exception: Exception in Constructor 29194312
finally
```

既に生成されたリソースの close が呼ばれる

NIO2

基本的な使い方

Files クラスに Path オブジェクトを渡すことで操作する。

■ FileSystem#getPath と Paths#get の違い

http://itpro.nikkeibp.co.jp/article/COLUMN/20110725/362803/?ST=develop

Paths クラスの get メソッドを使用すれば、FileSystem クラスの getPath と同じように Path オブジェクトを生成できます。ただし、Paths クラスではデフォルトのファイルシステムしか扱うことができません。

ディレクトリ探索

FileVisitor を walkFileTree へ渡すことでディレクトリ探索ができる

ディレクトリ監視